

# Document made available under the Patent Cooperation Treaty (PCT)

International application number: PCT/KR05/000668

International filing date: 09 March 2005 (09.03.2005)

Document type: Certified copy of priority document

Document details: Country/Office: KR  
Number: 10-2005-0007330  
Filing date: 24 January 2005 (24.01.2005)

Date of receipt at the International Bureau: 30 June 2005 (30.06.2005)

Remark: Priority document submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b)



World Intellectual Property Organization (WIPO) - Geneva, Switzerland  
Organisation Mondiale de la Propriété Intellectuelle (OMPI) - Genève, Suisse



별첨 사본은 아래 출원의 원본과 동일함을 증명함.

This is to certify that the following application annexed hereto  
is a true copy from the records of the Korean Intellectual  
Property Office

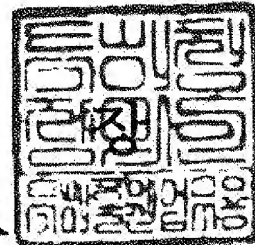
출 원 번 호 : 특허출원 2005년 제 0007330 호  
Application Number 10-2005-0007330

출 원 일 자 : 2005년 01월 24일  
Date of Application JAN 24, 2005

출 원 인 : 양세양  
Applicant(s) YANG, Sei Yang

2005 년 06 월 09 일

특 허 청  
COMMISSIONER



**【서지사항】**

<b>【서류명】</b>	특허출원서
<b>【권리구분】</b>	특허
<b>【수신처】</b>	특허청장
<b>【참조번호】</b>	0001
<b>【제출일자】</b>	2005.01.24
<b>【발명의 국문명칭】</b>	검증 성능과 검증 효율성을 높이는 동적검증 기법 방식의 검증 장치 및 이를 이용한 검증 방법론
<b>【발명의 영문명칭】</b>	Dynamic-Verification-Based Verification Apparatus Achieving High Verification Performance and Verification Efficiency, and the Verification Methodology Using the Same
<b>【출원인】</b>	
<b>【성명】</b>	양세양
<b>【출원인코드】</b>	4-1998-037998-4
<b>【발명자】</b>	
<b>【성명】</b>	양세양
<b>【출원인코드】</b>	4-1998-037998-4
<b>【우선권 주장】</b>	
<b>【출원국명】</b>	KR
<b>【출원종류】</b>	특허
<b>【출원번호】</b>	10-2004-0017476
<b>【출원일자】</b>	2004.03.09
<b>【증명서류】</b>	미첨부
<b>【우선권 주장】</b>	
<b>【출원국명】</b>	KR
<b>【출원종류】</b>	특허

【출원번호】	10-2004-0118529		
【출원일자】	2004. 12. 28		
【증명서류】	미첨부		
【취지】	특허법 제42조의 규정에 의하여 위와 같이 출원합니다. 출원인 양세양 (인)		
【수수료】			
【기본출원료】	0 면	38,000 원	
【가산출원료】	80 면	80,000 원	
【우선권주장료】	2 건	40,000 원	
【심사청구료】	0 항	0 원	
【합계】	158,000 원		
【감면사유】	개인(70%감면)		
【감면후 수수료】	75,400 원		
【첨부서류】	1. 요약서 · 명세서(도면)_2통		

## 【요약서】

### 【요약】

본 발명은 설계된 매우 복잡한 디지털 시스템의 설계 검증을 위한 시뮬레이션을 효율적으로 이용하는 검증 장치와 이를 이용한 효과적인 검증 방법에 관한 것이다.

본 발명에서는 임의의 컴퓨터에서 수행되어지는 본 발명의 검증 소프트웨어로 하여금 설계 코드에 부가적인 코드나 부가적인 회로를 추가하여 1회 이상의 시뮬레이션을 수행하게 한다. 시뮬레이션 수행은 앞단 시뮬레이션과 후단 시뮬레이션으로 나누어지며, 앞단 시뮬레이션의 결과를 효율적으로 이용하는 후단 시뮬레이션은 1 이상의 컴퓨터에서 수행되는 1 회 이상의 시뮬레이션을 1 이상의 시뮬레이터를 이용하여 순차적 수행을 가능하게 할뿐만 아니라, 네트워크 상에서 연결된 2 이상의 컴퓨터에서 돌아가는 2 이상의 시뮬레이터를 이용하여 상호 완전 독립적인 병렬적 수행도 가능하게 함으로서 전체 검증 시간과 검증 비용의 대폭적인 단축을 가능하게 하고, 검증의 효율성을 크게 높일 수 있게 한다.

### 【대표도】

도 1

## 【명세서】

### 【발명의 명칭】

검증 성능과 검증 효율성을 높이는 동적검증 기법 방식의 검증 장치 및 이를 이용한 검증 방법론 {Dynamic-Verification-Based Verification Apparatus Achieving High Verification Performance and Verification Efficiency, and the Verification Methodology Using the Same}

### 【도면의 간단한 설명】

- <1> 도1 은 본 발명에 관한 설계 검증 장치의 일 예를 개략적으로 도시한 도면.
- <2> 도2 는 본 발명에 관한 설계 검증 장치의 또 다른 일 예를 개략적으로 도시한 도면.
- <3> 도3 은 본 발명에서 제안하는 방법으로 시뮬레이션을 수행하는 과정을 개략적으로 도시한 도면.
- <4> 도4 는 본 발명에서 제안하는 앞단 시뮬레이션 단계인 1차 시뮬레이션과 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션을 통하여 설계 오류를 발견하고 수정하는 과정을 개략적으로 도시한 도면.
- <5> 도5(a) 은 도1 내지는 도2 와 같은 장치를 이용한 설계 검증을 수행하는 과정을 개략적으로 도시한 순서도를 도시한 도면.
- <6> 도5(b) 는 도1 내지는 도2 와 같은 장치를 이용한 설계 검증을 수행하는 또 다른 과정을 개략적으로 도시한 순서도를 도시한 도면.

<7> 도6(a) 는 도2 와 같은 장치를 이용한 설계 검증에서 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션을 병렬적으로 수행하는 과정을 개략적으로 도시한 도면.

<8> 도6(b) 는 도2 와 같은 장치를 이용한 설계 검증에서 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션을 병렬적으로 수행하는 또 다른 과정을 개략적으로 도시한 도면.

<9> 도6(c) 는 도2 와 같은 장치를 이용한 설계 검증에서 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션을 병렬적으로 수행하는 또 다른 과정을 개략적으로 도시한 도면.

<10> 도6(d) 는 도2 와 같은 장치를 이용한 설계 검증에서 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션을 병렬적으로 수행하는 또 다른 과정을 개략적으로 도시한 도면.

<11> <도면의 주요부분에 대한 부호의 설명>

<12>                    32 : 검증 소프트웨어                    34 : 시뮬레이터

<13> 35 : 컴퓨터

### 【발명의 상세한 설명】

### 【발명의 목적】

【발명이 속하는 기술분야 및 그 분야의 종래기술】

<14> 본 발명은 설계된 수백만 게이트급 이상의 디지털 시스템의 설계를 검증하는  
기술에 관한 것으로, 설계된 수백만 게이트급 이상의 디지털 시스템을 시뮬레이션

을 통하여 검증하고자 하는 경우에 시뮬레이션을 이용한 검증의 성능과 효율성을 증가시키는 검증 장치 및 이를 이용한 검증 방법에 관한 것이다.

<15>           최근에 집적회로의 설계 및 반도체 공정기술이 급격하게 발달함에 따라 디지털 회로 내지는 디지털 시스템 설계의 규모가 최소 수백만 게이트급에서 수천만 게이트급까지 커짐은 물론 그 구성이 극히 복잡해지고 있는 추세이고, 이와 같은 추세는 계속적으로 확대되고 있는 추세로 가까운 미래에 일억 게이트급 이상의 설계도 예상되고 있다. 그러나, 시장에서의 경쟁은 더욱 더 치열해지므로 빠른 시간 내에 우수한 제품을 개발하여야만 하는데 최근의 칩 설계에서 설계 검증은 전체 설계 시간에서 최대 70%까지의 시간을 차지하게 됨으로서, 빠른 시간 내에 자동화된 방법으로 설계된 회로를 효율적으로 설계 검증하기 위한 효과적인 방법의 필요성이 더욱 커지고 있다.

<16>           디지털 시스템을 칩으로 설계하는 경우에는 설계하는 대상이 두 가지가 있는데 그 하나는 DUV(Design Under Verification)이고, 또 다른 하나는 테스트벤치(testbench, 앞으로는 이를 TB로 약칭함)이다. DUV는 궁극적으로 반도체 제조 공정을 거쳐서 칩으로 만들어지는 설계 대상이고, 테스트벤치는 구현된 해당 칩이 장착되어서 동작하는 주변상황을 모델링한 것으로서 DUV의 시뮬레이션에 사용된다. DUV를 시뮬레이션을 통하여 검증하고자 하는 경우에 테스트벤치가 DUV에 입력을 인가하고 인가된 입력으로 DUV에서 출력되는 결과를 받아들이어서 처리하는 것이 일반적이다.

<17>           지금까지는 설계된 디지털 회로를 설계 검증하기 위하여서 하드웨어 기술언



어(Hardware Description Language, 앞으로 이를 HDL로 약칭함)나 시스템 기술 언어(System Description Language, 앞으로 이를 SDL로 약칭함)와 필요에 따라서는 하드웨어검증언어(Hardware Verification Language, 앞으로 이를 HVL로 약칭함)까지를 사용하거나, 또는 이들을 복합적으로 사용하고 있다. 설계 초기에는 소프트웨어적 접근법인 HDL 시뮬레이터들(예로, Verilog 시뮬레이터, VHDL 시뮬레이터, SystemVerilog 시뮬레이터 등)나 SDL 시뮬레이터(예로, SystemC 시뮬레이터, HW/SW co-simulator 등)이 주로 사용되어지고 있는데, 경우에 따라서는 이들과 더불어서 HVL 시뮬레이터들(예로, Vera 시뮬레이터, e 시뮬레이터 등)도 같이 사용되고 있다. 이와 같은 시뮬레이터는 설계 검증 대상회로와 테스트벤치를 소프트웨어적으로 모델링한 순차적인 인스트럭션 시퀀스로 구성된 소프트웨어 코드를 컴퓨터 상에서 순차적으로 수행하여야 함으로 상기 수백만 게이트급 이상의 설계에 대해서는 시뮬레이션 성능의 저하가 설계대상의 크기에 비례하여 발생하는 것이 문제가 되고 있다. 일 예로, 1000만 게이트급 이상의 설계를 HDL 시뮬레이터나 SDL 시뮬레이터로 시뮬레이션하는 경우에 현존하는 제일 빠른 프로세서를 장착한 컴퓨터에서 해당 HDL 시뮬레이터로 설계를 시뮬레이션하는 경우에 시뮬레이션 속도는 레지스터전송 수준(Register Transfer Level, 앞으로 이를 RTL로 약칭함)으로 하는 경우에 10-100 cycles/sec를 넘기기 어려우며, 게이트 수준에서 시뮬레이션을 진행하면 1-10 cycles/sec를 넘기기가 어려운 것이 매우 일반적이다. 그러나 해당 설계를 검증하고자 필요한 시뮬레이션 사이클은 최소 수백만 사이클에서부터 최대 수십억 사이클이 필요함으로 전체 시뮬레이션 시간은 상상을 초월하게 오래 걸리게 된다.

이와 같은 장시간의 검증 시간을 단축하기 위하여 현재에 사용되는 기술들은 다음과 같은 것들이 있는데, 첫째는 하드웨어 기반의 검증 시스템(예로, 시뮬레이션가속기, 하드웨어 에뮬레이터, FPGA 프로토타이핑 시스템 등)을 사용하거나, 둘째는 1 이상의 컴퓨터(예로, 100대의 워크스테이션)들 각각에 HDL 시뮬레이터를 인스톨하고 이를 고속의 네트워크를 통하여 연결한 시뮬레이션팜(simulation farm)을 사용하는 것이다. 그러나 하드웨어 기반의 검증 시스템을 사용하는 것은 설계 초기에는 적용이 불가능하고, 합성(synthesis)이나 컴파일 과정이 HDL 시뮬레이터를 사용하는 것보다 훨씬 오래 걸리며, 사용하기가 HDL 시뮬레이터에 비하여 매우 어렵고, 시스템의 구입 비용과 유지/보수 비용이 매우 클 뿐만 아니라, 무엇보다도 설계자나 검증엔지니어들이 HDL 시뮬레이터에 대한 선호도가 이들 하드웨어 기반의 검증 시스템에 비하여 매우 높고, HDL 시뮬레이터로는 아무 문제가 없이 수행이 되는 설계 코드들이 하드웨어 기반의 검증 시스템에서는 수행되지 않는 경우가 많아서 이들 하드웨어 기반의 검증 시스템들은 제한적인 상황과 제한적인 사용자들에서만 사용되고 있다. 또한 시뮬레이션팜을 이용하여 시뮬레이션의 성능을 향상시키는 것은 시뮬레이션을 위한 설계 코드나 테스트벤치가 2이상의 경우에만 가능할 뿐만 아니라, 테스트벤치들이 여러 개인 경우라고 하더라도 이들 중에서 제일 수행시간을 요하는 테스트벤치에 의하여 전체 시뮬레이션 시간이 결정되는 문제점(예를 든다면 특정 테스트벤치에 의한 시뮬레이션 시간이 일주일을 요하는 경우에는 시뮬레이션 팜(simulation farm)을 이용하더라도 일주일의 시뮬레이션 시간은 더 이상 단축이 안됨)이 있다. 이와 같은 상황은 SDL 시뮬레이터를 사용하는 경우도 마찬가지

이고, HVL 시뮬레이터를 사용하는 경우도 마찬가지이다.

<19>

특히, 최근에는 테스트벤치의 복잡도가 매우 증가하고 테스트벤치 내에 다양한 컴퍼넌트들(예로, 랜덤 입력스티뮬러스 발생기, 모니터, 검사기, 커버리지분석기, 응답검사기 등)이 존재함으로써 상위수준에서 추상화되어져 구술된 테스트벤치의 오버헤드가 매우 커짐으로 이것 또한 시뮬레이션의 수행 속도를 떨어뜨리는 주요 요소 중의 하나로 작용하고 있다. 이와 같은 다양한 테스트벤치 혹은 경우에 따라서는 DUV 내의 컴퍼넌트들은 최근의 발전된 검증(advanced verification) 기법에서 검증의 정량화와 자동화를 위하여 사용되는 테스트벤치 자동화(testbench automation), 커버리지지향 검증(coverage-driven verification), 주장기반 검증(assertion-based verification)등에서 반드시 필요한 것들이지만 앞서서 지적한대로 이들의 사용은 시뮬레이션의 속도를 더욱 떨어뜨리는 부정적인 부수효과를 초래하게 된다. 상황이 더욱 안좋게 되는 것은 이와 같은 발전된 검증 기법을 통하여서는 버그의 존재 여부만을 알 수 있거나, 기껏해야 버그가 존재하는 대략적인 위치만을 예측하는 것만이 가능하고, 버그가 정확하게 어디에 존재하고 있는지를 알아내는 것은 극히 어렵다는 것이다. 버그가 정확하게 어디에 존재하는지를 알아내고 이를 제거하는 것은 지능을 소유한 설계자나 검증엔지니어들의 몫이며, 이들 위해서는 시뮬레이션 과정에서 DUV에 그리고 경우에 따라서는 TB에 존재하는 시그널들이나 변수들의 값들을 탐침(probing)하여 저장하고, 이를 설계자나 검증엔지니어들이 확인하는 과정이 반드시 필요하게 된다. 그러나, 이와 같은 두 가지 상황들이 합쳐지면서 시뮬레이션의 속도 저하는 더욱 크게 나타나게 된다. 더욱이 별도의

HVL을 사용하여서 테스트벤치를 구술하게되면 일반적으로 HDL 시뮬레이터들의 API(Application Program Interface)인 VPI/PLI/FLI 등을 반드시 사용하여야 하는데 이것의 사용도 시뮬레이션의 수행 속도를 떨어뜨리는 주요 요소 중의 또 다른 하나로 작용하고 있다.

<20> 뿐만 아니라, 지금까지의 설계는 레지스터전송수준(Register Transfer Level, 앞으로는 이를 RTL로 약칭함)에서의 설계에서 합성 기술을 사용하여서 네트리스트를 자동 생성하는 설계 방법이 주류를 이루고 있다. 한편에서는 설계 복잡도의 증가를 해결하고자 RTL 수준보다도 높은 단계인 행위수준(behavioral level)내지는 시스템레벨수준에서 DUV와 TB 모두를 구술하여서 RTL 구조를 자동 생성하고 최종적으로 네트리스트도 자동 생성하고자 하는 시도들이 있다. 그러나, 이와 같은 새로운 방법이 보편화될 수 있을지는 매우 불투명한 상황인데, 이유로는 대부분의 하드웨어 설계자들은 RTL 수준에서의 설계에 매우 익숙해져 있기 때문이며, RTL 수준보다 상위 수준에서 구술된 설계의 합성 결과가 일반적으로 RTL에서 구술되어 합성된 결과에 비하여 동작속도/면적 등에서 많이 뒤떨어지기 때문이다. 그러나 최근의 수천만 게이트급의 설계는 대부분 내부에 1개 이상의 프로세서를 내장하고 있는 SOC(System On a Chip) 설계인데, 이와 같은 SOC들은 내장된 프로세서를 구동하는 대용량의 소프트웨어를 내장하고 있어서 SOC 설계에서는 이와같은 내장 소프트웨어의 개발이 하드웨어의 개발과 동시에적으로 진행되어질 수 있는 하드웨어/소프트웨어 동시설계(co-design) 및 동시검증(co-verification)이 필수적이다. 그러나 하드웨어 설계자들이 절대적으로 선호하는 RTL 수준에서 구술된 DUV의 수행속도는 앞서서

언급된 것과 같이 수행 속도가 너무 느림으로 내장 소프트웨어를 개발하기 위한 플랫폼으로는 사용할 수가 없는 반면에, 소프트웨어 개발자들은 개발하고자 하는 내장 소프트웨어들이 신속하게 수행될 수 있는 플랫폼을 하드웨어 설계가 진행되고 있는 시점에 이미 필요로 하고 있는 문제점이 있다.

<21>

현재 제일 많이 사용되어지고 있는 시뮬레이션은 이벤트-구동(event-driven) 시뮬레이션이다. 시뮬레이션은 이 외에 사이클-기반(cycle-based) 시뮬레이션, 트랜잭션-기반(transaction-based) 시뮬레이션 등이 있다. 추상화(abstraction)된 기준에서 본다면 이벤트-구동이 사이클-기반보다는 덜 추상화된 것이고 사이클-기반이 트랜잭션-기반 보다는 덜 추상화된 것이다. 사이클-기반 시뮬레이션은 이벤트-구동 시뮬레이션보다 대략 10-100배 정도 빠르지만 이벤트-구동 시뮬레이션에 비하여 제약성이 많아서 현재에는 하드웨어 검증에는 거의 사용되지 못하고 소프트웨어 검증의 가상 플랫폼(virtual platform)으로만 사용되고 있는 실정이다. 즉, 사이클-기반 시뮬레이션을 이용하게 되면 이벤트-구동 시뮬레이션에 비하여 훨씬 빠르게 시뮬레이션을 수행하는 것이 가능하지만, 극히 제한적인 상황에서만 이용가능하고 이의 시뮬레이션 결과도 전적으로 신뢰할 수 없음으로, 현재 HDL을 이용하는 하드웨어 설계자들로부터는 사이클-기반 시뮬레이션은 철저히 외면되어지고 있으며 하드웨어 설계자들의 대부분은 이벤트-구동 시뮬레이션을 절대적으로 선호하고 있는 실정이다. 또한 현재 이벤트-구동 시뮬레이션, 사이클-기반 시뮬레이션, 트랜잭션-기반 시뮬레이션 등은 각기 독립적으로만 사용되고 있는 상황이다. 결과적으로, 이와 같은 상황들은 전체 검증의 효율성과 성능을 제약하는 요소가 되고 있다.

## 【발명이 이루고자 하는 기술적 과제】

<22>

따라서, 본 발명의 목적은 초대규모급 디지털 시스템 설계에 대한 검증을 위한 시뮬레이션의 성능 및 효율성을 크게 향상시키는 시뮬레이션 기반의 설계 검증 장치 및 이를 이용한 설계 검증 방법을 제공함에 있다. 특히 시뮬레이션을 수행한 후에 설계 오류에 대한 디버깅을 수행하기 위하여 설계 코드에 존재하는 시그널들이나 변수들에 대한 가시도(visibility)가 요구되는데, 문제는 설계 오류의 정확한 위치를 알아내기 위해서 시뮬레이션 수행 전에 구체적으로 어느 특정 시그널들이나 변수들에 대한 가시성이 어느 시뮬레이션 구간에서 필요한지를 예측하기 어렵다는 것이다. 따라서 시뮬레이션을 수행할 때에 처음부터 설계 코드에 존재하는 모든 시그널들과 변수들에 대하여 탐침이 가능하게 이들 모두를 덤프(dump) 대상으로 선택한 다음에 시뮬레이션을 수행한다. 그러나 설계 코드에 존재하는 모든 시그널들과 변수들을 덤프하면서 시뮬레이션을 수행하는 경우에는, 덤프를 전혀하지 않고 시뮬레이션을 수행하는 것과 비교하여 시뮬레이션 수행시간이 대략 2배에서 많게는 10배 이상 길어지게 된다. 본 발명에서는 설계 코드에서 버그의 위치를 정확히 파악하기 위하여 설계 코드에 존재하는 모든 시그널들과 변수들에 대하여 시뮬레이션 처음부터 덤프를 수행하는 기존의 방법과는 다르게 시뮬레이션 수행시간을 크게 늘리지 않고서 내지는 기존의 방법과 비교하여 시뮬레이션 수행시간을 크게 단축하면서도 설계 코드에서 버그의 위치를 찾을 수 있도록 하는 자동화된 방법과 이를 위한 검증장치를 제공함을 목적으로 한다.

<23>

본 발명의 또 다른 목적으로는, 발전된 검증 기법에서 사용되는 테스트벤치

자동화, 커버리지지향 검증, 주장기반 검증 등을 사용하면서도, 동시에 DUV에 대한 높은 가시도를 신속하게 확보하는 자동화된 방법과 이를 위한 검증 장치를 제공하는 것이다.

<24>           본 발명의 또 다른 목적으로는, 하향식으로 진행되는 설계 과정에 있어서 추상화 단계의 상위 수준에서 수행된 시뮬레이션 결과를 추상화 단계의 하위 수준에서 수행되는 시뮬레이션에 이용함으로써 DUV에 대한 높은 가시도를 신속하게 확보하는 동시에 전체 시뮬레이션 시간을 크게 단축할 수 있도록 하는 자동화된 방법과, 이를 위한 검증 장치를 제공함으로써 하드웨어 검증 내지는 소프트웨어 검증 내지는 하드웨어/소프트웨어 동시 검증을 효과적으로 가능하게 하는 것이다.

<25>           본 발명의 또 다른 목적으로는, 상위 단계에서 하위 단계에서 진행되는 설계 과정에 따라서 검증도 상위 단계에서 하위 단계로 진행되는 과정에서 상위 단계의 검증 결과를 하위 단계의 검증 결과에 자동화된 방식으로 사용하게 하고, 상위 단계의 검증 결과를 활용하여 하위 단계의 검증을 신속하게 수행될 수 있게 할 뿐만 아니라, 필요에 따라서는 상위 단계의 검증 결과를 레퍼런스로 하여서 하위 단계의 검증을 효율적으로 수행할 수 있도록 함으로서 여러 추상화 단계들에 걸쳐서 존재하고 있는 전체적인 검증의 속도 향상과 효율성 향상을 가능하게 하는 것이다.

<26>           본 발명의 또 다른 목적으로는, 상위 단계에서 하위 단계에서 진행되는 설계 과정에 따라서 검증도 상위 단계에서 하위 단계로 진행되는 과정에서 상위 단계의 검증 결과를 하위 단계의 검증 결과에 자동화된 방식으로 사용하게 하고, 상위 단계의 검증 결과를 활용하여 하위 단계의 검증을 신속하게 수행될 수 있게 할 뿐만

아니라, 필요에 따라서는 하위 단계의 검증 결과를 레퍼런스로 하여서 상위 단계의 검증을 효율적으로 수행할 수 있도록 함으로서 여러 추상화 단계들에 걸쳐서 존재하고 있는 전체적인 검증의 속도 향상과 효율성 향상을 가능하게 하는 것이다.

<27>        본 발명의 또 다른 목적으로는, 상위 단계에서 하위 단계에서 진행되는 설계 과정에 따라서 검증도 상위 단계에서 하위 단계로 진행되는 과정에서 각 단계에 제일 적합한 방식의 시뮬레이션을 트랜잭션-기반 시뮬레이션, 사이클-기반 시뮬레이션, 이벤트-구동 시뮬레이션 중에서 최적하게 선정하여 사용하게 하고, 특정 추상화 단계의 검증 결과를 다른 추상화 단계의 검증 결과에 자동화된 방식으로 사용하게 하여서 상기 다른 추상화 단계에서의 검증을 신속하게 수행될 수 있게 하여서 전체적인 검증의 속도향상과 효율성 향상을 가능하게 하는 것이다.

### 【발명의 구성】

<28>        상기 목적들을 달성하기 위하여, 본 발명에 따른 설계 검증 장치는 검증 소프트웨어와 1 이상의 시뮬레이터가 인스톨된 1 이상의 컴퓨터로 구성된다. 검증 소프트웨어는 컴퓨터에서 실행되며, 만일 상기 설계 검증 장치에 2 이상의 컴퓨터들이 있는 경우에는 이들 2 이상의 컴퓨터는 네트워크로 연결되어져서 컴퓨터들 간에 파일들의 이동을 네트워크를 통하여 가능하게 한다. 상기 1 이상의 시뮬레이터는 이벤트-구동 시뮬레이터로만 구성될 수도 있고, 혹은 이벤트-구동 시뮬레이터와 사이클-기반 시뮬레이터로 같이 구성될 수도 있고, 혹은 사이클-기반 시뮬레이터로만 구성될 수도 있고, 혹은 사이클-기반 시뮬레이터와 트랜잭션-기반 시뮬레이터로 같이 구성될 수도 있고, 혹은 이벤트-구동 시뮬레이터와 사이클-기반 시뮬레이터와



트랜잭션-기반 시뮬레이터로 같이 구성될 수도 있다.

<29>

본 발명에서 제안되는 검증 장치와 검증 방법은 하드웨어 설계 코드 자체를 검증하는 함수적 검증(functional verification)에 사용될 수 있을 뿐만 아니라, 설계코드를 합성한 게이트수준의 네트리스트를 이용한 게이트수준의 검증에서도 사용될 수 있고, 또는 배치(placement) 및 배선(routing)이 되고 추출된 타이밍정보를 게이트수준의 네트리스트에 첨부시켜(back-annotated) 수행하는 타이밍 검증에서도 사용될 수 있다. 그러나 앞으로의 설명은 설계 코드 자체를 검증하는 함수적 검증에 대하여 하기로 하며, 게이트수준 검증이나 타이밍 검증에 대해서도 같은 방법을 적용할 수 있음으로 구체적인 설명은 생략하기로 한다. 뿐만 아니라, 함수적 검증에서도 레지스터전송수준(Register Transfer Level, 앞으로 이를 RTL로 약칭함)에서 뿐만아니라, 이보다 상위 수준인 행위수준(behavioral level)이나 트랜잭션수준(transaction level)에서도 적용할 수 있는데 앞으로의 설명은 주로 RTL에서의 함수적 검증에 대하여 설명하고, 다른 수준들에서의 함수적 검증의 구체적인 설명은 생략하기로 한다. 뿐만 아니라, 이들 여러 수준들에서의 검증들을 통합화하여 진행되는 혼용 수준에서의 검증에서도 적용되어질 수 있다.

<30>

상기 검증 소프트웨어는 설계 코드를 읽은 후에 여기에다 추가적으로 부가 코드나 부가 회로를 자동화된 방식으로 부가한다. 부가 코드나 부가 회로는 기본적으로 추가되어지는 HDL 코드 내지는 SDL 코드 내지는 HVL 코드 내지는 C/C++ 코드 내지는 시뮬레이션 명령어 내지는 이들의 조합들로서, 시뮬레이션을 수행하는 과정에서 시뮬레이션 시간 상에서 일정 간격이나 혹은 1 이상의 특정 시뮬레이션 시

점들에서 시뮬레이션의 상태나, 가시도가 필요한 DUV에서의 1 이상의 설계객체(추후에 설명됨)들의 상태와 경우에 따라서는 TB의 상태까지를 저장하는 역할을 수행하고, 또한 추후에 사용자의 요구에 따라서 저장된 시뮬레이션의 상태나, 저장된 상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태로부터 내지는 저장된 상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태와 TB의 상태로부터 다시 시뮬레이션을 재개하게 한다. 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태는 저장하지만, TB의 상태를 저장하지 않는 경우에는 이 대신에 시뮬레이션 전 구간에 걸쳐서 상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 모든 입력 값들과 입력모드시의 모든 입출력 값들을 이 입력들에서나 입출력들에서 매 이벤트 발생시마다 내지는 매 사이클마다 내지는 매 트랜잭션마다 저장하는 것이 필요하다. 시뮬레이션의 상태란 시뮬레이터라는 소프트웨어 프로그램이 수행되는 과정에서의 특정 시뮬레이션 시점에서 시뮬레이터의 모든 정적 정보와 동적 정보를 일컫는 말이다. 이는 멀티프로그래밍 환경에서 임의의 프로그램(혹은 프로세스 내지는 쓰레드)이 잠시 수행을 멈추는 상태(wait)로 갔다가 나중에 다시 재개(resume)하기 위해서 저장이 되어져야 하는 프로그램의 상태 정보와 유사하다. 설계객체의 상태(state)란 설계객체에 존재하는 변수나 시그널들의 특정 시뮬레이션 시점에서의 값들을 일컫는 말로서, 특정 설계객체의 상태 정보는 완전상태 정보와 불완전상태 정보로 나눌 수 있고 최소상태 정보는 불완전상태 정보의 특별한 경우이다. 특정 설계객체의 완전상태란 해당 설계객체에 존재하는 모든 변수나 시그널들의 특정 시뮬레이션 시점에서의 값들을 일컫는 말이고 설계객체의 불완전상태란

해당 설계객체에 존재하는 1 이상의 임의의 변수나 시그널들의 특정 시물레이션 시점에서의 값들을 일컫는 말이다. 또한 설계객체의 최소상태란 해당 설계객체에 존재하는 모든 변수나 시그널들 중에서 기억소자(기억소자란 플립플롭 혹은 래치 혹은 메모리 셀을 말함)의 출력 이름이 되는 변수나 시그널들만의 값들과 조합궤환루프(combination feed-back)가 상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들에 존재하는 경우에는 궤환루프 경로상의 이름이 되는 변수나 시그널들만의 값을 모두 합친 것을 일컫는 말이다. TB의 상태란 TB에 존재하는 변수나 시그널들의 특정 시물레이션 시점에서의 값들을 일컫는 말이다. 또한 이와 같은 저장된 시물레이션의 상태나, 저장된 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태와 경우에 따라서는 TB의 상태로부터(가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태는 저장하지만, TB의 상태를 저장하지 않는 경우에는 TB의 상태 대신에 시물레이션 전 구간에 걸쳐서 상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 모든 입력 값들과 입력모드시의 모든 입출력 값들을 이 입력들에서나 입출력들에서 매 이벤트 발생시마다 내지는 매 사이클마다 내지는 매 트랜잭션마다 저장된 것과 상기 저장된 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태를 이용하여) 다시 시물레이션을 재개하는 경우에는 사용자의 의도에 따라서 설계 코드에 존재하는 모든 시그널들과 변수들에 대한 탐침을 수행하기 위한 덤프를 병행하거나 혹은 설계 코드에 존재하는 특정 시그널들이나 변수들에 대한 탐침을 수행하기 위한 덤프를 병행한다. 시물레이터로 HDL 시물레이터(예로, Verilog 시물레이터나 VHDL 시물레이터나 SystemVerilog 시물레이터 등)를 사용하는 경우에 시물레이

선의 상태를 저장하는 방법의 한 예로서 HDL 시뮬레이터의 save(NC-Verilog, Verilog-XL, VCS의 경우)나 혹은 checkpoint(ModelSim의 경우)라는 명령을 사용하고, 저장된 특정 시뮬레이션 상태에서부터 시뮬레이션을 특정 시간에서부터 재개하는 방법의 한 예로서 HDL 시뮬레이터의 restore(ModelSim) 혹은 restart(NC-Verilog, Verilog-XL, VCS)와 같은 명령을 사용할 수 있다. 또한 1 이상의 특정 설계객체의 저장된 상태를 특정 시뮬레이션 시작 시에 해당 상기 1 이상의 특정 설계객체의 초기상태로서 사용하여서 시뮬레이션을 하기 위해서는 상기 1 이상의 특정 설계객체 내에 존재하는 해당 변수들이나 시그널들이 상기 저장된 상태의 값들을 가지고 시뮬레이션을 시작할 수 있도록 하면 된다. 이를 위해서는 시뮬레이터에서 제공하는 다양한 제어능력(controllability) 방법들을 사용할 수 있는데, 일 예들을 든다면 시뮬레이터의 XMR(Cross Module Reference)을 통하던지 혹은 force/release procedural statement 혹은 PLI/VPI/FLI와 같은 시스템테스크 등을 사용할 수 있다. 따라서, 본 검증 장치의 검증소프트웨어를 통하여서 상기의 부가 코드나 부가 회로가 원 설계 코드에 부가되어져서 이와 같은 기능 들을 자동적으로 수행할 수 있도록 한다. 이와 같은 시뮬레이션 방법이 검증의 성능 향상 및 효율성을 증대시킬 수 있는 이유는 다음과 같다. 이미 언급된 대로 시뮬레이션을 통하여 검증을 수행하는 과정에서는 설계 오류를 발견하고 이를 수행하는 과정에서 반드시 설계 코드에 존재하는 특정 시그널들이나 변수들의 값들을 특정 시간대에서 알 수 있도록 하는 탐침(probing)하는 과정이 항상 필요하다. 그러나 문제는 이와 같은 설계 코드에서 설계 오류를 발견하고 이를 수정하기 위해서 탐침이 필요한 특정 시그널

들이나 변수들이 어떤 것인지를 시뮬레이션 수행 전에 정확히 예측할 수가 없을 뿐만 아니라, 이들 탐침 대상의 탐침 시점이 언제 필요한지도 시뮬레이션 수행 전에 정확히 예측할 수가 없다. 따라서 시뮬레이션을 1차 적으로 수행한 후에 이 1차 시뮬레이션 결과를 바탕으로 설계 오류의 위치를 파악하기 위하여 필요한 특정 시뮬레이션 시점에서의 특정 시그널들이나 변수들을 탐침 대상으로 선정한 후에 2차 시뮬레이션을 시뮬레이션 시간 0에서부터 1차 시뮬레이션 종료시점까지 진행하면서 탐침 대상이 된 시그널들이나 변수들을 특정 시간대에서 덤프를 수행하게 된다. 이와 같이 2차 시뮬레이션 과정에서도 설계 오류의 위치를 파악하지 못하면, 새로운 시그널들이나 변수들을 탐침 대상으로 선정하고 시뮬레이션을 다시 시뮬레이션 시간 0에서부터 반복하게 되며, 이와 같은 과정을 설계 오류의 위치가 발견되기까지 수 차례 반복하는 과정이 필요하다. 그러나 이와 같은 반복적인 시뮬레이션을 시뮬레이션 시간 0에서부터 2회 이상 반복하게 됨으로서 전체 검증 시간은 크게 늘어나게 된다. 만일 이와 같은 반복적인 시뮬레이션을 피하고자 하는 경우에는 제일 처음 수행하는 시뮬레이션 수행 시에 설계 코드에 존재하는 모든 시그널들과 변수들을 탐침 대상으로 설정하고 시뮬레이션 전과정에 걸쳐서 이들을 덤프하면서 진행하여야 한다. 그러나 이와 같이 설계코드에 존재하는 모든 시그널들과 변수들을 덤프하면서 시뮬레이션을 진행하게 되면 덤프를 진행하지 않고 시뮬레이션을 진행하는 것에 비하여 시뮬레이션 시간이 대략 2배에서부터 최대 10배 이상 증가하게 되어 이 또한 전체 검증 시간을 크게 늘리게 된다. 뿐만 아니라 이와 같이 설계 코드에 존재하는 모든 시그널들과 변수들을 전체 시뮬레이션 구간에 걸쳐서 덤프하게 되면

덤프되는 시뮬레이션 파형(simulation waveform) 데이터의 크기가 수십 기가바이트  
 에서부터 수백 기가바이트 이상 증가하게 된다. 이와 같은 큰 용량의 시뮬레이션  
 파형 데이터를 저장하기 위해서는 대용량의 하드 디스크를 필요로 할 뿐만 아니라,  
 이와 같이 특정 형식(예로, Verilog의 VCD/extended-VCD 형식 내지는 Novas사의  
 FSDB 형식 내지는 SHM, VCS+와 같은 특정 시뮬레이터 벤더들의 압축된 파형형식)으  
 로 하드디스크에 저장된 시뮬레이션 파형 데이터를 컴퓨터로 읽어와서 파형분석기  
 (waveform viewer)로 분석하는 과정에서도 매우 긴 시간이 필요하게 되어 이 또한  
 전체 검증 시간을 증가시키게 된다. 본 특허에서 제안되는 시뮬레이션 방법은 시뮬  
 레이션의 구성을 앞단 시뮬레이션 단계와 후단 시뮬레이션 단계의 2 단계로 나누어  
 서 수행되도록 한다. 앞단 시뮬레이션 단계에서는 1차 시뮬레이션 수행을 설계 코  
 드에 존재하는 모든 시그널들과 변수들을 덤프하지는 않도록 함으로서 시뮬레이션  
 시간의 증가를 초래하지 않고 신속하게 시뮬레이션을 진행할 수 있게 한다. 이와  
 같은 1차 시뮬레이션 과정에서 수행되어지는 것은 후단 시뮬레이션 단계인 1차 이  
 후의 추가 시뮬레이션들을 기존 방법에서와 같은 시뮬레이션 시간 0에서 수행되지  
 않고 사용자가 관심이 있는 시뮬레이션 시간대에서 아주 가까운 곳에서부터 시작할  
 수 있도록 시뮬레이션 일정 간격(예로, 시뮬레이션 시작에서부터 매 100,000 나노  
 초마다, 혹은 매 50,000 시뮬레이션 사이클마다)마다 혹은 원하는 시뮬레이션 시점  
 들  $t_0, t_1, \dots, t_t$ 마다 시뮬레이션의 상태나, 가시도가 필요한 DUV에서의 1 이상의  
 설계객체들의 상태와 경우에 따라서는 TB의 상태  $s_0, s_1, \dots, s_t$ 를 저장(가시도가

필요한 DUV에서의 1 이상의 설계객체들의 상태는 저장하지만 TB의 상태를 저장하지 않는 경우에는 TB의 상태 저장 대신에 시뮬레이션 전 구간에 걸쳐서 상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 모든 입력 값들과 입력모드시의 모든 입출력 값들을 이 입력들에서나 입출력들에서 매 이벤트 발생시마다 내지는 매 사이클 마다 내지는 매 트랜잭션 마다 저장)시키는 것이다. 1 이상의 특정 시뮬레이션 시점에서 설계객체들의 상태나 입력/출력/입출력에서의 값들의 저장은 dump 명령 (예로, PLI 시스템 테스트인 \$dumpvars 내지는 \$dumpport 등) 등을 통하여 수행되어진다. 이와 같이 1 이상의 시뮬레이션 시점들에서 시뮬레이션의 상태나, 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태와 경우에 따라서는 TB의 상태를 저장(가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태는 저장하지만, TB의 상태를 저장하지 않는 경우에는 TB의 상태 저장 대신에 시뮬레이션 전 구간에 걸쳐서 상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 모든 입력 값들과 입력 모드시의 모든 입출력 값들을 이 입력들에서나 입출력들에서 매 이벤트 발생시마다 내지는 매 사이클마다 내지는 매 트랜잭션마다 저장)하게 되면, 이후부터는 재 시뮬레이션을 시뮬레이션 시간 0에서뿐만 아니라 이 각 시뮬레이션 시점들  $t_0$ ,  $t_1$ , ...  $t_i$ 의 하나에서부터도 가능하게 된다. 따라서 앞단 시뮬레이션 단계인 1차 시뮬레이션을 통하여 설계 코드(설계 코드란 DUV 내지는 TB를 구술한 코드)에 존재하는 시그널들이나 변수들 중에서 설계 오류의 위치를 찾기 위하여 탐침이 필요한 특정 시그널들이나 변수들을 선정(필요한 경우에는 설계 코드에 존재하는 모든 시그널들

과 변수들을 선정)하고 이들 탐침 대상에 대한 탐침이 어느 시뮬레이션 시간대에서 필요한가를 파악한 다음(파악하는 방법의 예로는 로깅을 하거나 주장기법(assertion technique)을 사용하는 것임)에, 후단 시뮬레이션 단계에서 수행되는 1차 이후의 시뮬레이션들에서는 앞단 시뮬레이션 단계인 1차 시뮬레이션에서 저장된 1 이상의 시뮬레이션 상태나, 저장된 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태 저장과 경우에 따라서는 TB의 상태들에서 이 저장(가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태는 저장하지만, TB의 상태를 저장하지 않는 경우에는 TB의 상태 저장 대신에 시뮬레이션 전 구간에 걸쳐서 상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 모든 입력 값들과 입력모드시의 모든 입출력 값들을 이 입력들에서나 입출력들에서 매 이벤트 발생시마다 내지는 매 사이클마다 내지는 매 트랜잭션마다 저장)이 일어난 시뮬레이션 시간이 탐침이 필요한 시뮬레이션 시간대  $(t_s, t_e)$ 의 시작시점  $t_s$ 에서 시뮬레이션 시간적으로 앞서면서(시뮬레이션 시간 10,000 나노초와 20,000 나노초가 있는 경우에, 10,000 나노초는 20,000 나노초에 비하여 시간적으로 앞서 있다고 하고, 20,000 나노초는 10,000 나노초에 비하여 시간적으로 뒤에 있다고 함) 제일 가까운 곳에 있는 시간대로 시뮬레이션 시간을 재설정하기 위하여 해당 시점의 시뮬레이션 상태 내지는 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 저장된 상태와 TB의 상태까지 저장된 경우는 TB의 저장된 상태까지를 갖는 상태  $S_i$ 로 시뮬레이션의 상태나 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태(TB의 상태를 저장하지 않은 경우에는 상기 가시



도가 필요한 DUV에서의 1 이상의 설계객체들의 상태만을 이용함)와 TB의 상태까지 저장된 경우는 TB의 저장된 상태 각각으로 상기 1 이상의 설계객체들과 TB의 상태를 설정하고 시뮬레이션 시간  $t_i$ 에서부터  $t_e$ 까지 시뮬레이션을 수행하면서 해당 탐침 대상에 대한 탐침을 진행하고, 필요시에는  $S_i$ 보다 시간적으로 앞선 시뮬레이션 상태나 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 저장된 상태와 경우에 따라서는 저장된 TB의 상태들  $S_{i-1}$ ,  $S_{i-2}$ , ...,  $S_{i-n}$  각각으로 시뮬레이션의 상태나 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태(TB의 상태를 저장하지 않은 경우에는 상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태만을 이용함)와 TB의 상태까지 저장된 경우는 TB의 저장된 상태 각각으로 상기 1 이상의 설계객체들과 TB의 상태를 순차적으 설정하면서 각각의 시뮬레이션 시간  $t_{i-1}$ ,  $t_{i-2}$ , ...,  $t_{i-n}$ 에서부터  $t_i$ ,  $t_{i-1}$ , ...,  $t_{i-n+1}$ 까지 순서적으로 추가적인 n번의 시뮬레이션을 순차적으로 설계 코드에서 설계 오류의 위치와 원인이 밝혀 질 때까지 진행한다. 따라서 이와 같은 과정을 통하여 설계에 대한 디버깅을 신속하게 할 수 있는데, 이와 같은 과정은 기존의 시뮬레이션 기반의 검증 방법들에 비하여 검증 시간을 크게 단축시켜 줄 수 있으므로 매우 효율적인 검증 방법이 된다. 이와 같은 시뮬레이션 방법을 A 시뮬레이션 방법이라고 칭한다.

<31>        또 다른 방법으로서는 설계 코드에서 DUV나 테스트벤치에 존재하는 계층구조를 이용한 분할 및 정복(divide & conquer) 방법을 사용할 수 있다. 이 경우에는

검증 소프트웨어가 설계 코드를 읽어 들어서 DUV와 테스트벤치에 대하여 분할을 수행하여 설계 코드를 2 이상의 설계블록들로(M개의 설계블록들로 분할되었다고 가정) 나누고, 이 설계블록 각각에 존재하는 모든 입력과 입출력들을 탐침 대상으로 선정한 후에 이들 탐침 대상들이 앞단 시뮬레이션 단계인 1차 시뮬레이션 과정에서 덤프되어질 수 있도록 설계 코드에 부가 코드나 부가 회로를 부가한다. 여기서 설계블럭(design block)들이란 크게 DUV 뿐만 아니라 테스트벤치 모두를 포함한다. 뿐만 아니라, 이들 DUV와 테스트벤치는 일반적으로 계층적인 구조로 내부에 다양한 1 이상의 하위모듈들을 가지고 있는데, 이들 하위모듈 각각도 설계블럭이라고 할 수 있다. 이와 같은 설계블럭들이나 DUV, 그리고 테스트벤치 각각이나 이들의 조합들을 모두 설계객체(design object)들이라고 할 수 있다. 따라서 DUV 전체가 하나로서 설계객체가 될 수 있을 뿐만 아니라, DUV 내부에 존재하는 특정 설계블럭 하나도 설계객체가 될 수도 있고, 2 이상의 설계블럭들이 모여서 하나의 설계객체가 될 수도 있다. 앞단 시뮬레이션 단계인 1차 시뮬레이션을 수행하면서 각 설계블럭들의 모든 입력과 입출력들을 덤프하여 파일 형태로 저장한 후에, 이를 본 발명의 검증 소프트웨어를 이용하여 해당 설계블럭들에 대한 M개의 테스트벤치들로 변환하여 해당 설계블럭의 설계 코드와 같이 해당 테스트벤치를 같이 시뮬레이션 컴파일을 수행하여 M개의 설계블럭들에 대응되는 M개의 시뮬레이션 실행파일을 생성한다. 이와 같이 각 설계블럭별로 새로운 테스트벤치를 생성하여 시뮬레이션 컴파일을 수행하여 별도의 시뮬레이션 실행파일을 생성하게 되는 것에 비하여, 해당 설계블럭들의 모든 입력들과 입출력들을 탐침된 파일(예로, VCD/extended VCD 파일

내지는 FSDB 파일)을 테스트벤치화 하지 않고 직접 이용하게 되면 변환 시간과 시뮬레이션 컴파일 시간을 절약할 수 있지만 해당 설계블록을 추후에 시뮬레이션 하는 경우에 필요한 입력값들을 상기 탐침된 파일에서 읽어서 인가해주기 위하여서는 API(Application Program Interface)를 사용하는 것(Verilog에서는 VPI/PLI, VHDL에서는 FLI 등)이 필요하다. 그러나 이와 같은 API를 사용하여 별도의 코드를 시뮬레이터에 연동시키게 되면 API 오버헤드로 인하여 시뮬레이션의 속도가 크게 저하됨으로 득보다 실이 많을 수 있다. 그러나 최근에 사용되는 특정 시뮬레이터(예로, Synopsys의 VCS 7.0이상)에서는 API를 사용하지 않고 외부 코드모듈(예로, C, C++ 코드)를 시뮬레이터에 연동시키는 것이 가능하기도 한데, 이와 같은 경우에는 직접 앞단 시뮬레이션 단계인 1차 시뮬레이션에서 덤프를 통하여 얻어진 탐침 파일을 직접 이용하여 시뮬레이션 컴파일을 수행하여 M개의 시뮬레이션 실행 파일을 생성하는 것도 가능하다. 이 후에 설계 오류에 대한 위치를 파악하기 위하여 추가적인 시그널들이나 변수들에 대한 탐침이 필요한 경우에는 상기 M개의 시뮬레이션 실행파일들 중에서 해당 시그널들이나 변수들이 존재하는 해당 설계블록과 상기 1차 시뮬레이션 후에 생성된 덤프 파일이나 테스트벤치로써 시뮬레이션 컴파일하여 얻어진 특정 시뮬레이션 실행파일을 이용한 추가적인 시뮬레이션을 하게 된다. 이와 같은 추가적인 시뮬레이션에서는 해당 설계블록들에 존재하는 모든 시그널들이나 변수들에 대한 탐침을 수행하면서 시뮬레이션을 진행하거나, 또는 관심대상이 되는 특정 시그널들이나 변수들에 대한 탐침을 수행하면서 시뮬레이션을 진행한다. 만일, 탐침이 필요한 특정 시그널들이나 변수들이 2 이상의 설계블록들에 분산되어져서 존

재하는 경우에는 이들 설계블록들에 대하여 상기 설계블록이 하나인 과정과 동일한 과정들을 순서적으로 반복하여 수행하여 각 과정에서 얻어진 탐침 결과들을 통합하면 된다. 이와 같이 원래의 DUV와 원래의 복잡한 테스트벤치 전체를 시뮬레이션하지 않고서 특정 설계블록들과 덤프 파일이나 새롭게 생성된 테스트벤치만을 이용한 시뮬레이션을 수행하게 되면 시뮬레이션 속도를 크게 증가시킬 수 있다. 이와 같은 시뮬레이션 방법은 B 시뮬레이션 방법이라 칭한다. 또한 B 시뮬레이션 방법에서 앞단 시뮬레이션 단계인 1차 시뮬레이션에서 VPI/PLI/FLI 등을 사용하여 시뮬레이션을 진행하면서 VCD 내지는 FSDB를 생성하는 대신에 집적 1 이상의 테스트벤치 파일을 생성시키는 것도 가능한데, 이와 같은 경우에는 1차 시뮬레이션 후에 VCD 내지는 FSDB를 테스트벤치로 변환시키는 과정을 생략 가능하다.

<32>        이와 같은 두 가지 방법들은 기존의 시뮬레이션 방법들에 비하여 설계객체들에 대한 높은 가시도를 제공하면서도 시뮬레이션의 속도를 크게 저하시키지 않을뿐만 아니라, 앞단 시뮬레이션과 후단 시뮬레이션을 다른 추상화된 수준들에서 수행하게 되면 별도의 특별한 하드웨어 기반의 검증 플랫폼들(예로, 하드웨어 에뮬레이터 또는 FPGA 프로토타이핑 플랫폼 등)을 사용하지 않고서 시뮬레이션의 속도를 크게 향상시키는 것도 가능하다 (자세한 내용은 아래에서 설명됨). 물론 상기 두 가지 방법들인 A 시뮬레이션 방법과 B 시뮬레이션 방법을 병행한 검증 방법도 가능하다.

<33>        그러나 위에서 언급된 A 시뮬레이션 방법과 B 시뮬레이션 방법의 경우에 다음과 같은 문제점들이 있을 수 있다. 우선 A 시뮬레이션 방법의 경우에 후단 시뮬

레이션 단계인 2차 이후의 시뮬레이션에서 시뮬레이션 하고자 하는 시간대 ( $t_s, t_e$ )가 매우 긴 경우에  $t_i, t_{i-1}, t_{i-2}, \dots, t_{i-n}$  시점에서의 총  $n+1$  횟수의 시뮬레이션을 순차적으로 수행하여야 함으로서 원래 시뮬레이션 방식보다는 검증 시간을 단축할 수 있지만 그래도 많은 검증 시간이 A 시뮬레이션 방법에 필요하게 된다. 또한 B 시뮬레이션 방법의 경우에도 2차 이후의 시뮬레이션에서 시뮬레이션 하여야 하는 설계블록들의 숫자가 큰 경우에는 이들을 순차적으로 수행하여야 함으로서 원래 시뮬레이션 방식보다는 검증 시간을 단축할 수 있지만 그래도 많은 검증 시간이 B 시뮬레이션 방법에도 필요하게 된다. 그러나, 시뮬레이터가 2 이상이고 이들 시뮬레이터들이 수행되는 복수개의 컴퓨터들(예로, X개의 시뮬레이터가 X개의 컴퓨터에 인스톨되어져 있음)이 네트워크로 연결되어져 있는 경우에는 이와 같은 A 시뮬레이션 방법이나 B 시뮬레이션 방법에서 수행되는 후단 시뮬레이션 단계인 1차 이후의 추가 시뮬레이션들을 동시에 병렬적으로 수행하는 것이 가능하다. 이와 같은 병렬적 시뮬레이션은 이 병렬적으로 수행되어져야 하는 각 시뮬레이션들이 완전히 독립적으로 수행하는 것이 가능함으로서 후단 시뮬레이션 단계인 1차 이후의 추가 시뮬레이션의 수행을 획기적으로 빠르게 수행하는 것이 가능하다. 그런데 A 시뮬레이션 방법에서의 병렬적 수행은 시간적으로 병렬성을 얻는 것이므로 시간적 병렬수행(temporally parallel execution)이라고 하고, B 시뮬레이션 방법에서의 병렬적 수행은 공간적으로 병렬성을 얻는 것이므로 공간적 병렬수행(spatially parallel execution)이라 칭하기로 한다.

앞단 시뮬레이션 단계인 1차 시뮬레이션의 목적과 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션의 목적을 비교하여 보면, 1차 이후의 시뮬레이션을 통하여 DUV와 경우에 따라서는 TB까지에 대한 가시도를 얻는 것이 필요함으로 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션에서 많은 수행시간이 필요함을 알 수가 있는데, 본 특허의 방법을 이를 상기의 시간 병렬과 공간 병렬 방법에 의하여 크게 줄일 수 있다. 하지만, 전체적인 시뮬레이션의 속도 향상을 위해서는 앞단 시뮬레이션 단계인 1차 시뮬레이션까지도 가능한 최대한도로 빠르게 수행하면서, 1차 이후와 1회 이상의 시뮬레이션을 수행하는데 필요한 정보를 수집하는 것이 매우 중요하다. 이를 위하여서는 여러 가지 방법들을 생각할 수 있다. 그 중에서 첫째 방법은 앞단 시뮬레이션 단계인 1차 시뮬레이션도 2 이상의 컴퓨터에서 돌아가는 2 이상의 시뮬레이터를 사용하여 병렬적으로 수행하는 것이다. 그러나, 이와 같은 앞단 시뮬레이션 단계인 1차 시뮬레이션에서의 병렬 수행은 각 시뮬레이터가 독립적으로 수행되는 것이 아니고 서로 연동되면서 수행하여야 함으로 많은 통신 오버헤드와 동기화 오버헤드가 발생할 수 있다. 둘째 방법은 앞단 시뮬레이션 단계인 1차의 시뮬레이션을 빠른 시뮬레이션이 수행될 수 있도록 원래의 설계 코드 대신에 원래 설계 코드와 함수적으로 등가이나 다른 구문 형식으로 변환된 설계 코드를 사용하여 시뮬레이션하는 것이다. 이는 원 설계 코드에서 시뮬레이션의 수행을 오래 걸리게 하는 구문이 포함되어 있는 경우에 이 구문과 함수적으로 등가이며 시뮬레이션의 수행을 빠르게 하는 다른 구문으로 변환시키는 것인데, 구체적인 예를 든다면 원 설계 코드에 루프(loop)문들을 펼침으로(un-rolling) 루프문을 제거하거나

감지 리스트(sensitivity list)에서 불필요한 리스트들을 제거하거나, 경우에 따라서는 지연관련 구문을 제거하는 것 등이다. 이와 같은 변환 과정은 본 특허에서의 검증 장치의 검증 소프트웨어를 이용하여 자동적으로 수행될 수 있다. 셋째 방법은 앞단 시뮬레이션 단계인 1차 시뮬레이션에서는 후단 시뮬레이션 단계에서 사용하는 것보다 추상화 단계를 높여서 시뮬레이션을 수행하고, 후단 시뮬레이션 단계에서 상기 높은 추상화 단계에서 수행된 시뮬레이션 결과를 활용하여서 1차 이후의 시뮬레이션을 신속하게 수행하는 것이다. 구체적인 예를 든다면, 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션을 이벤트-구동 시뮬레이터를 사용하는 경우에 앞단 시뮬레이션 단계인 1차 시뮬레이션에서는 이벤트-구동(event-driven) 시뮬레이터를 사용하는 대신에 수행속도 면에서 이벤트-구동 시뮬레이터보다 대략 10-100배 정도가 빠른 사이클-기반(cycle-based) 시뮬레이터를 사용하는 것이다. 셋째 방법을 위해서 Verilog나 VHDL로 코딩된 설계 코드를 자동화된 방법(예로, HDL2SystemC 변환툴을 사용하여서 자동 변환)이나 수동으로 SystemC 코드로 변환시켜서 HDL 시뮬레이터 대신에 SystemC 시뮬레이터를 사용할 수 있다. 일반적으로 SystemC 시뮬레이터는 이벤트스케줄러가 필요 없음으로 Verilog 시뮬레이터나 VHDL 시뮬레이터와 비교하여 빠르게 수행하는 것이 가능하다. 둘째와 셋째 방법의 경우에는 앞단 시뮬레이션 단계인 1차 시뮬레이션을 SystemC 시뮬레이터나 사이클-기반(cycle-based) 시뮬레이터로써 수행하면서 시뮬레이션의 상태를 주기적 간격이나 1 이상의 특정 시뮬레이션 시점들에서 1회 이상 저장하는 대신에, 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태 저장과, 경우에 따라서는 TB의 상태까지를 주기적 간격이나

1회 이상의 특정 시뮬레이션 시점들에서 1회 이상 저장하고(가시도가 필요한 DUV에서의 1 이상의 설계객체들이 상태는 저장하지만, TB의 상태를 저장하지 않는 경우에는 TB의 상태 저장 대신에 시뮬레이션 전 구간에 걸쳐서 상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 모든 입력 값들과 입력모드시의 모든 입출력 값들을 이 입력들에서나 입출력들에서 매 이벤트 발생시마다 내지는 매 사이클마다 내지는 매 트랜잭션마다 저장하고), 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션을 2 이상의 컴퓨터 상에서 돌아가는 2 이상의 HDL 시뮬레이터를 사용하고 병렬적으로 수행하게 된다. 이를 위해서 상기 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션 각각을 상기 앞단 시뮬레이션 단계인 1차 시뮬레이션 수행 과정에서 1회 이상의 시뮬레이션 시점들에서 저장된 1 이상의 설계 코드 상태 정보들 각각으로 초기상태를 다르게 설정하여서 수행한다. 후단 시뮬레이션 단계에서의 1차 이후에 수행되는 시뮬레이터에 사용되는 테스트벤치는 앞단 시뮬레이션 단계에서의 1차 시뮬레이션에서 사용한 테스트벤치(예로, SystemC로 구술된 테스트벤치)를 그대로 사용할 수 있을 뿐만 아니라, 둘째와 셋째 방법에서는 1차 시뮬레이션에서 사용되는 시뮬레이터와 1차 이후 시뮬레이션에서 사용되는 시뮬레이터가 서로 상이함으로 시뮬레이션 상태를 이용하는 것이 불가능함으로 1차 시뮬레이션 과정에서 저장한 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태정보를 이용하여야 함과 더불어서 1차 시뮬레이션 과정에서 TB의 상태를 저장하지 않은 경우에는 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 입력값과 입력 모드에서의 입출력값들도 1차 시뮬레이션 전 과정에 걸쳐서 매 이벤트 발생마다 내지



는 매 사이클마다 내지는 매 트랜잭션마다 지속적으로 탐침하여 저장하고(예로, VCD dump나 FSDB dump를 함), 이를 테스트벤치로 변환해서 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션에서 사용할 수도 있다. 이와 같이 원래의 테스트벤치를 사용하지 않고 앞단 시뮬레이션 단계인 1차 시뮬레이션에서 생성된 테스트벤치를 사용하게 되면, 테스트벤치의 파일 사이즈가 커짐으로 컴파일 시간은 늘어나게 되지만 테스트벤치의 구성이 단순 패턴기반 형태가 됨으로 원래의 복잡한 테스트벤치가 시뮬레이션 런타임 시에 소비하는 시뮬레이션 시간에서 테스트벤치가 소비하는 시간을 단축할 수 있는 장점이 있다.

<35> 뿐만 아니라, 위에서 언급한 기법은 RTL 설계 코드를 이용하여 RTL에서 시뮬레이션한 결과를 RTL 설계 코드를 합성하여서 게이트-네트리스트를 생성하고, 필요 시에는 배치 및 배선 결과에 의한 지연시간 정보까지를 가져와서 수행되는 게이트 수준에서의 시뮬레이션 결과에 이용함으로써 게이트수준에서의 시뮬레이션 시에서의 디버깅을 매우 효과적으로 하는 것에 이용할 수 있다. 즉, 앞단 시뮬레이션 단계인 1차 시뮬레이션을 RTL 설계 코드를 이용하여 RTL에서 이벤트-구동(event-driven) 방식 내지는 사이클-기반(cycle-based) 방식 시뮬레이션하면서 설계코드에 대한 상태정보를 1 이상의 특정 시뮬레이션 시점들에서 저장하고, 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션은 게이트수준에서의 시뮬레이션을 상기 RTL 시뮬레이션에서 상기 1 이상의 특정 시뮬레이션 시점들에서 저장된 1 이상의 상태정보들 각각을 이용함으로써 1 이상의 게이트수준에서의 시뮬레이션들을 병렬적으로 수행하는 것도 가능하다. 상기 게이트수준에서의 시뮬레이션은

SDF(Standard Delay Format) 등을 이용하여서 타이밍 시뮬레이션도 가능하고, 타이밍 정보를 이용하지 않는 게이트수준의 함수적 시뮬레이션도 가능하다.

<36> 위에서 설명된 것과 같이, 앞단 시뮬레이션 단계인 1차 시뮬레이션을 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션 보다 좀 더 추상화된 수준에서 수행하면서 가시도가 필요한 DUV에서의 1 이상의 설계객체들에 대한 동적정보(예로, 설계검증대상의 최소상태정보와 필요시에는 입력정보도 포함, 혹은 설계검증대상의 완전 상태정보)를 수집하고, 상기 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션에서 상기 수집된 동적정보를 이용하는 경우 전체 시뮬레이션의 속도 향상 이외의 또 다른 매우 중요한 장점은 다음과 같다. 현재의 설계 방법은 하향식 설계기법(top-down design methodology)을 많이 이용하는데, 이와 같은 하향식 설계에서 상위에서 하위로(예로, 행위단계에서 레지스터전송수준을 거쳐서 게이트수준으로) 내려오면서 설계가 진행되어오는 것에 맞추어서 검증도 상위에서 하위로(예로, 행위단계에서 레지스터전송수준을 거쳐서 게이트수준으로) 내려오면서 수행되어야 한다. 따라서, 위에서 언급된 방법은 상위 단계에서 얻어진 시뮬레이션 결과를 이 보다 하위 단계(예로, 행위수준에서의 시뮬레이션 결과를 이 보다 하위 단계인 레지스터전송수준에서, 또는 레지스터전송수준에서의 시뮬레이션 결과를 이 보다 하위 단계인 게이트수준)에서 수행되는 시뮬레이션에 적용할 수 있도록 함으로서 하향식 설계의 전 과정에서 설계의 일관성을 검증 과정에서 체계적이며 효율적으로 점검할 수 있게 하는 매우 중요한 추가적인 장점이 있다. 이와 같은 시뮬레이션 방법은 새로운 검증 방법론(verification methodology)이라고 할 수 있는데, 추상화가 더 된 설계

코드를 이용한 검증 결과를 추상화가 될 된 설계코드를 이용한 검증에 사용함으로써 전체 검증 시간과 디버깅 시간을 크게 줄일 수 있을 뿐만 아니라, 전체 검증 시간과 전체 설계 시간까지도 크게 줄일 수 있는 새로운 방법이다. 이는 설계 구현 단계(implementation phase)에서 Magma Design Automation 사가 물리적 합성 (physical synthesis)에서 사용하여 큰 성공을 거둔 기술인 이득-기반의 합성 (GBS:Gain-based Synthesis) 기술에서 얻을 수 있는 장점과 유사한 장점을 검증 과정에서 얻을 수 있게 한다. 즉 현재 반도체 설계 과정에서의 최대 문제점은 추상화를 통하여서 대규모의 설계를 효율적으로 할 수 있게 된 장점의 부정적인 부수효과로서 설계 과정이 진행되면서 앞 단계에서 예측된 결과와는 다른 결과가 후 단계에서 나타나게 되고, 이는 설계 과정에서 과도한 설계 및 검증 반복(design & verification iteration)을 초래한다는 것이다. 그런데 본 특허에서의 방법과 GBS 방법은 상위단계에서 하위단계로 진행되면서 변화가 되는 상황에 최대한도의 일관성을 제공함으로써 모두 이와 같은 과도한 반복을 최대한 방지시킬 수 있는 방법이라는 공통점이 있다. 차이점으로는 두 기술이 하나는 검증 관련 기술이고 하나는 구현 기술이라는 것과, 또 다른 개념적 차이는 GBS에서는 추상화 단계에서 하위 수준의 정보(타이밍 정보)에 맞추어지도록 상위 수준에서부터 이에 맞추어 나가는 것인 반면에, 본 특허에서 검증에서 적용하는 기술은 대부분의 경우에 해당하는 정상적 상황에서는 이와는 정 반대로 추상화 단계에서 정해진 상위 수준의 검증 정보에 맞추어지도록 하위수준에서 검증 및 디버깅을 진행하는 것이라 할 수 있다. 즉, 상위 수준의 검증 결과가 하위 수준 검증에서의 레퍼런스 역할(또는 골든 역할)을 수

행함으로서 전체 검증 및 설계 과정을 신속하게 수행할 수 있게 된다. 이와 같은 2 이상의 추상화 수준들에서의 시뮬레이션 결과들을 통합하여서 상위 수준에서의 시뮬레이션 결과를 하위 수준에서의 시뮬레이션에 이용할 수 있게 하는 것은 많은 장점들이 있다. 상위 수준에서의 시뮬레이션이 되는 대상이 DUV 전체가 되는 것이 성능 면에서 제일 바람직하지만, 설계 재사용(design reuse) 등과 같은 이유 등에서 DUV의 1 이상의 특정 설계객체들만이 상위 수준으로 구술된 설계에 대해서도 위와 같은 개념은 그대로 적용되어질 수 있다. 예로, DUV 내에 5개의 설계블록 B0, B1, B2, B3, B4가 존재하고 B0와 B1은 이벤트-구동 Verilog RTL 코드로 구술되어 있고 B2, B3, B4는 사이클-기반 SystemC 코드로 구술되어 있는 경우에는 B0과 B1에 대한 이벤트-구동 방식의 시뮬레이션이 B2와 B3와 B4에 대한 사이클-기반의 시뮬레이션과 연동되어져서, DUV 부분적으로는 상위 수준에서 시뮬레이션이 수행됨으로 시뮬레이션의 속도를 높이는 것도 가능하다. 일반적으로 추상화가 더 된 상위수준에서의 시뮬레이션 결과는 추상화가 덜 된 하위수준에서 DUV가 모델링되어져서 모의 동작하는 많은 경우들의 특정한 경우라고 생각할 수 있다. 따라서 이와 같은 특정한 경우에도 DUV는 반드시 올바른 동작을 보장하여야 하며, 이와 같은 사실에서부터 상위수준에서의 시뮬레이션 결과는 대부분의 경우에 매우 유용한 레퍼런스의 역할을 수행할 수 있게 됨을 알 수 있다.

<37> 뿐만 아니라 위에서 설명된 것과 유사하게, 2 이상의 추상화 수준들에서의 시뮬레이션 결과들을 통합하여서 하위 수준에서의 시뮬레이션 결과를 상위 수준에서의 시뮬레이션에 이용할 수 있게 하는 것도 가능하다.

<38> 뿐만 아니라, 위에서 언급된 방법들을 통하여 수집된 시뮬레이션 결과들(예로, 시뮬레이션 과정에서 DUV와 TB에서 덤프된 VCD나 FSDB)을 1 이상의 특정설계 오류가 디버깅 과정을 통하여 수정된 이후에 이 특정 설계 오류가 실제로 제거가 되었는지 그리고 이 디버깅 과정에서 실수로 다른 설계 오류가 도입되지 않았는지를 조사하기 위한 확인 시뮬레이션에서 이용하여 이 확인 시뮬레이션이 매우 신속하게 이루어질 수 있게 하는 장점이 있다. 이와 같은 확인 시뮬레이션의 신속한 수행방식을 점진적 시뮬레이션(incremental simulation) 방식이라고 하는데, 점진적 시뮬레이션에 대한 구체적인 방법인 별도의 특허 문서(대한민국 특허출원번호 10-2004-55330와 10-2004-93310)에 자세히 기술되어 있으므로 여기에서 구체적인 설명은 생략하기로 한다.

<39> 상기 목적 외에 본 발명의 다른 목적 및 이점들은 첨부한 도면을 참조한 실시 예에 대한 상세한 설명을 통하여 명백하게 드러나게 될 것이다.

<40> 도1 은, 컴퓨터에서 운영되는 본 발명의 검증 소프트웨어와 시뮬레이터를 갖는 컴퓨터로 구성된 본 발명에 관한 설계 검증 장치의 일 예를 개략적으로 도시한 도면이다.

<41> 도2 는, 컴퓨터에서 운영되는 본 발명의 검증 소프트웨어와 2 이상의 시뮬레이터를 갖는 2 이상의 컴퓨터들과 이들 컴퓨터들을 연결하는 컴퓨터 네트워크로 구성된 본 발명에 관한 설계 검증 장치의 또 다른 일 예를 개략적으로 도시한 도면이다.

<42> 도3 은 앞단 시뮬레이션 단계인 1차 시뮬레이션 수행 도중에 주기적으로 내

지는 1 이상의 특정 시뮬레이션 시점들에서 시뮬레이션의 상태나, 저장된 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태와 경우에 따라서는 TB의 상태를 저장(가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태는 저장하지만, TB의 상태를 저장하지 않는 경우에는 TB의 상태 저장 대신에 시뮬레이션 전 구간에 걸쳐서 상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 모든 입력 값들과 입력모드시의 모든 입출력 값들을 이 입력들에서나 입출력들에서 매 이벤트 발생시마다 내지는 매 사이클마다 내지는 매 트랜잭션마다 저장)하고, 후단 시뮬레이션 단계인 2차 시뮬레이션 수행에서 1차 시뮬레이션에서 저장된 총  $n+1$ 개의 시뮬레이션 상태나, 저장된 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태와 경우에 따라서는 TB의 상태 중에서 특정 상태  $s_{i-1}$ 로 시뮬레이션을 설정하여(가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태는 저장하지만, TB의 상태를 저장하지 않는 경우에는 TB의 상태 대신에 시뮬레이션 전 구간에 걸쳐서 상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 모든 입력 값들과 입력모드시의 모든 입출력 값들을 이 입력들에서나 입출력들에서 매 이벤트 발생시마다 내지는 매 사이클마다 내지는 매 트랜잭션마다 저장된 것과 상기 저장된 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태  $s_{i-1}$ 를 이용하여서)  $t_{i-1}$ 에서부터  $t_i$ 까지 시뮬레이션을 수행하는 과정을 개략적으로 도시한 것이다.

<43>            도4 는 앞단 시뮬레이션 단계인 1차 시뮬레이션 수행 도중에 주기적으로 시뮬레이션의 상태나, 저장된 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상

태와 경우에 따라서는 TB의 상태를 저장(가시도가 필요한 DUV에서의 1 이상의 설계 객체들의 상태는 저장하지만, TB의 상태를 저장하지 않는 경우에는 TB의 상태 저장 대신에 시뮬레이션 전 구간에 걸쳐서 상기 가시도가 필요한 DUV에서의 1 이상의 설계 객체들의 모든 입력 값들과 입력모드시의 모든 입출력 값들을 이 입력들에서나 입출력들에서 매 이벤트 발생시마다 내지는 매 사이클마다 내지는 매 트랜잭션마다 저장)하고, 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션에서 상기 앞단 시뮬레이션 단계인 1차 시뮬레이션에서 저장된 1 이상의 시뮬레이션 상태나, 저장된 가시도가 필요한 DUV에서의 1 이상의 설계 객체들의 상태들과 경우에 따라서는 저장된 TB의 상태들에서(가시도가 필요한 DUV에서의 1 이상의 설계 객체들의 상태는 저장하지만, TB의 상태를 저장하지 않는 경우에는 가시도가 필요한 DUV에서의 1 이상의 설계 객체들의 저장된 상태들에서) 이 저장이 일어난 시뮬레이션 시간이 탐침이 필요한 시뮬레이션 시간대  $(t_s, t_e)$ 의 시작시점  $t_s$ 에서 시뮬레이션 시간적으로 앞서면서 제일 가까운 곳에 있는 시뮬레이션 상태나, 가시도가 필요한 DUV에서의 1 이상의 설계 객체들의 저장된 상태와 경우에 따라서는 TB의 저장된 상태  $S_i$ (가시도가 필요한 DUV에서의 1 이상의 설계 객체들의 상태는 저장하지만, TB의 상태를 저장하지 않는 경우에는 가시도가 필요한 DUV에서의 1 이상의 설계 객체들의 저장된 상태  $S_i$ )로 시뮬레이션의 상태나, 가시도가 필요한 DUV에서의 1 이상의 설계 객체들의 상태와 TB의 상태를 설정하고(가시도가 필요한 DUV에서의 1 이상의 설계 객체들의 상태는 저장하지만, TB의 상태를 저장하지 않는 경우에는 가시도가 필요한 DUV에서의 1

이상의 설계객체들의 상태를 설정하고) 시뮬레이션 시간  $t_i$ 에서부터  $t_e$ 까지 시뮬레이션을 수행하면서 해당 탐침 대상에 대한 탐침을 진행하고, 필요시에는  $S_i$ 보다 시간적으로 앞선 시뮬레이션 상태나, 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태와 경우에 따라서는 TB의 상태들  $S_{i-1}$ ,  $S_{i-2}$ , ...,  $S_{i-n}$  각각으로(가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태는 저장하지만, TB의 상태를 저장하지 않는 경우에는, 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태들  $S_{i-1}$ ,  $S_{i-2}$ , ...,  $S_{i-n}$  각각으로) 시뮬레이션의 상태나, 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 저장된 상태와 경우에 따라서는 TB의 저장된 상태를(가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태는 저장하지만, TB의 상태를 저장하지 않는 경우에는, 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 저장된 상태를) 순차적을 설정하면서 각각의 시뮬레이션 시간  $t_{i-1}$ ,  $t_{i-2}$ , ...,  $t_{i-n}$ 에서부터  $t_i$ ,  $t_{i-1}$ , ...,  $t_{i-n+1}$ 까지 순서적으로 추가적인 n번의 시뮬레이션을 순차적으로 진행하는 과정을 개략적으로 도시한 것이다.

<44> 도5(a) 은 도1 내지는 도2와 같은 장치를 이용한 설계 검증을 수행하는 과정을 개략적으로 도시한 순서도이다.

<45> 제 100 단계에서, 1차 시뮬레이션 수행 도중에 일정 간격이나 특정 시점에서 시뮬레이션의 상태 내지는 탐침대상이 되는 DUV내의 1 이상의 설계객체의 상태와 TB의 상태 내지는 탐침대상이 되는 DUV내의 1 이상의 설계객체의 상태가 자동적으



로 저장되도록 설계 코드에 추가 코드를 부가하고, 제 102 단계로 진행한다. 제 102 단계에서는, 상태 저장 간격이나 저장 시점 설정하고 제 104 단계로 진행한다. 제 104 단계에서는, 1차 시뮬레이션 수행과 동시에 상태 정보를 저장하고 제 106 단계로 진행한다. 제 106 단계에서는 탐침이 필요한 시뮬레이션 시간대 ( $t_s$ ,  $t_e$ ) 설정하고 제 108 단계로 진행한다. 제 108 단계에서는 1차 시뮬레이션 과정에서 저장된 상태 정보를 이용하여 시뮬레이션을 수행하면서 탐침 실행하고 제 110 단계로 진행한다. 제 110 단계에서는 설계 코드에 존재하는 설계 오류 발견 및 제거되면 제 112 단계로 진행하고, 그렇지 않으면 제 106 단계로 진행한다. 제 112 단계에서 설계 검증 완료가 완료된 것이 확인되면 전체 과정을 종료하게 되고, 그렇지 않으면 제 102 단계로 진행한다.

<46> 도5(b) 는 도1 내지는 도2와 같은 장치를 이용한 설계 검증을 수행하는 또 다른 과정을 개략적으로 도시한 순서도이다.

<47> 제 200 단계에서 설계 코드에 대한 분할을 수행 2 이상의 설계블럭들을 생성하고, 제 202 단계로 진행한다. 제 202 단계에서는 1차 시뮬레이션 수행 도중에 모든 설계블럭들의 입력과 입출력을 시뮬레이션 전구간에서 탐침하여 저장할 수 있도록 설계 코드에 추가 코드를 부가하고 제 204 단계로 진행한다. 제 204 단계에서는 1차 시뮬레이션 수행과 동시에 각 설계블럭들의 입력과 입출력에 대한 탐침 정보들을 저장한 후, 해당 설계블럭들과 같이 시뮬레이션 컴파일하여 1 이상의 시뮬레이션 실행 파일 생성하고 제 206 단계로 진행한다. 제 206 단계에서는 추가 탐침이

필요한 시그널들이나 변수들 설정하고 제 208 단계로 진행한다. 제 208 단계에서는 추가 탐침이 필요한 시그널들이나 변수들을 포함하는 1 이상의 설계블럭들을 포함하고 있는 1 이상의 시뮬레이션 이진 파일들을 이용한 시뮬레이션 수행 하며 해당 시그널들이나 변수들에 대한 탐침 수행을 수행하고 제 210 단계로 진행한다. 제 210 단계에서는 설계 코드에 존재하는 설계 오류 발견 및 제거하면 제 212 단계로 진행하고, 그렇지 못하면 제 206 단계로 진행한다. 제 212 단계에서는 설계 검증 완료가 확인되면 전체 과정을 종료하고, 그렇지 못하면 제 204 단계로 진행한다.

<48>

도6(a) 는 도2 와 같은 장치를 이용한 설계 검증에서 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션 과정에서 앞단 시뮬레이션 단계인 1차 시뮬레이션 과정에서 저장된 2 이상의 시뮬레이션 상태나, 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 저장된 상태와 TB의 상태까지 저장된 경우에는 TB 상태까지를 포함하는 상태(가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태는 저장하지만, TB의 상태를 저장하지 않는 경우에는, 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 저장된 상태)들 중에서 2 이상의 시뮬레이션 상태나, 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 저장된 상태와 TB의 상태까지 저장된 경우에는 TB의 상태까지를 포함하는 상태(가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태는 저장하지만, TB의 상태를 저장하지 않는 경우에는, 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 저장된 상태)로부터 2 이상의 시뮬레이터들을 사용하여 시뮬레이션을 시작할 수 있도록 설정하여 이 2 이상의 시뮬레이션을 병렬적으로 수행하고 이들 시뮬레이션 결과들을 취합하여 사용자에게 제공하는 과정을 도시한 그림이

다.

<49>

도6(b) 는 도2 와 같은 장치를 이용한 설계 검증에서 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션과정에서 앞단 시뮬레이션 단계인 1차 시뮬레이션 과정에서 저장된 2 이상의 설계 코드(DUV 전체나 DUV내의 1 이상의 설계객체를 가르키거나, 경우에 따라서는 DUV에 TB를 합친 것을 가르킴)의 상태정보들 중에서 2 이상의 상태정보들을 선택하여 이들 2 이상의 설계 코드의 상태정보들 각각으로 2 이상의 시뮬레이터들에서 병렬적으로 수행되는 설계 코드들 각각의 초기 상태를 설정하여 2 이상의 시뮬레이션을 병렬적으로 수행하고 이들 시뮬레이션 결과들을 취합하여 사용자에게 제공하는 과정을 도시한 것인데, 상기 앞단 시뮬레이션 단계인 1차 시뮬레이션을 RTL 시뮬레이션으로 수행하고 상기 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션을 게이트수준 시뮬레이션으로 수행하는 과정을 도시한 그림이다.

<50>

도6(c) 는 도2 와 같은 장치를 이용한 설계 검증에서 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션 과정에서 앞단 시뮬레이션 단계인 1차 시뮬레이션 과정에서 저장된 2 이상의 설계 코드(DUV 전체나 DUV내의 1 이상의 설계객체를 가르키거나, 경우에 따라서는 DUV에 TB를 합친 것을 가르킴)의 상태정보들 중에서 2이상의 상태정보들을 선택하여 이들 2 이상의 설계 코드의 상태정보들 각각으로 2이상의 시뮬레이터들에서 병렬적으로 수행되는 설계 코드들 각각의 초기 상태를 설정하여 이 2 이상의 시뮬레이션을 병렬적으로 수행하고 이들 시뮬레이션 결과들을 취합하여 사용자에게 제공하는 과정을 도시한 것인데, 상기 앞단 시뮬레이션 단계인 1차 시뮬레이션을 사이클-기반 시뮬레이션으로 수행하고 상기 후단 시뮬레이션 단계

인 1차 이후의 시뮬레이션을 이벤트-구동 RTL 시뮬레이션으로 수행하는 과정을 도시한 그림이다.

<51> 도2 와 같은 장치를 이용한 설계 검증에서 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션과정에서 앞단 시뮬레이션 단계인 1차 시뮬레이션 과정에서 설계 코드에 존재하는 1 이상의 특정 설계블록들의 입력과 입출력값들을 시뮬레이션 전과정에서 탐침하여 얻어진 2 이상의 덤프 파일들이나 이들을 테스트벤치로 변환된 파일들을 해당 설계블록들과 같이 2 이상의 시뮬레이터들을 이용하여 동시에 시뮬레이션 컴파일하여 2 이상의 시뮬레이션 실행파일들을 생성하여 이를 2 이상의 컴퓨터들에서 병렬적으로 실행시키고 이들 시뮬레이션 결과들을 취합하여 사용자에게 제공할 수 있다. 그러나 이와 같은 방법은 상기 앞단 시뮬레이션 단계인 1차 시뮬레이션 과정에서 설계 코드에 존재하는 1 이상의 특정 설계블록들의 입력과 입출력값들을 시뮬레이션 전과정에서 탐침하게 됨으로 시뮬레이션의 속도가 크게 떨어질 수 있다. 이를 개선하는 방법을 도6(d)로써 설명한다.

<52> 도6(d) 는 도2 와 같은 장치를 이용한 설계 검증에서 앞단 시뮬레이션 단계인 1차 시뮬레이션 과정에서 2 이상의 시뮬레이션 시점들에서 2 이상의 시뮬레이션 상태나, 저장된 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태와 경우에 따라서는 TB의 상태를 저장하고(가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태는 저장하지만, TB의 상태를 저장하지 않는 경우에는 TB의 상태 저장 대신에 시뮬레이션 전 구간에 걸쳐서 상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 모든 입력 값들과 입력모드시의 모든 입출력 값들을 이 입력들에서나 입출력

들에서 매 이벤트 마다 내지는 매 사이클 마다 내지는 매 트랜잭션 마다 발생시마다 저장하고), 후단 시뮬레이션 단계인 1차 이후에 처음으로 수행되는 시뮬레이션에서 2 이상의 컴퓨터에서 수행되는 2 이상의 시뮬레이터를 이용하여 상기 전단 시뮬레이션 단계인 1차 시뮬레이션 과정에서 저장된 2 이상의 시뮬레이션 상태나, 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 저장된 상태와 TB 상태까지도 저장된 경우에는 TB 상태까지 합한 상태로부터 시뮬레이션이 병렬적으로 진행할 수 있도록 하고, 이 병렬적 시뮬레이션이 진행되는 과정에서 설계 코드에 존재하는 1 이상의 특정 설계블록들의 입력과 입출력값들을 병렬적으로 탐침하여 얻어진 2 이상의 덤프 파일들이나 이들을 테스트벤치로 변환된 파일들을 해당 설계블록들과 같이 2 이상의 시뮬레이터들을 이용하여 동시에 시뮬레이션 컴파일하여 2 이상의 시뮬레이션 실행파일들을 생성하고, 상기 2 이상의 컴퓨터들에서 병렬적으로 시뮬레이션 하는 과정을 도시한 그림이다.

### 【발명의 효과】

<53> 상술한 바와 같이, 본 발명에 따른 검증 장치 및 이를 이용한 설계 검증 방법의 목적은 초대규모급 설계 검증을 위하여 시뮬레이션을 수행하는 경우에 앞단 시뮬레이션 단계인 1차 시뮬레이션을 통해서 후단 시뮬레이션 단계인 1차 이후에 추가적으로 진행될 시뮬레이션들을 매우 빠르고 효과적으로 진행하는 것에 필요한 최소한의 정보들을 수집하고, 이 후에 진행되는 추가적인 시뮬레이션들을 하나의 시뮬레이터를 이용하여 순차적으로, 내지는 2 이상의 시뮬레이터를 이용하여 병렬적으로 빠르게 진행하게 함으로서 전체의 시뮬레이션 시간을 단축하며, 빠른 시간 내

에 설계 코드에 존재하는 오류들의 위치를 정확히 찾아내어 수정하는 것이 가능함으로서 전체 설계 검증의 시간을 대폭적으로 단축하고 검증의 효율성을 크게 높이는 것이 가능하다.

<54>           이상 설명한 내용을 통해 당업자라면 본 발명의 기술사상을 일탈하지 아니하는 범위에서 다양한 변경 및 수정이 가능함을 알 수 있을 것이다. 따라서, 본 발명의 기술적 범위는 실시 예에 기재된 내용으로 한정되는 것이 아니라 특허 청구의 범위에 의하여 정하여져야만 한다.

## 【특허청구범위】

### 【청구항 1】

검증 소프트웨어와 1 이상의 시뮬레이터를 구비하는 설계검증 장치에 있어서,

상기 검증 소프트웨어는 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 자동화된 방식을 통하여 부가 코드나 부가 회로를 추가하여 앞단 시뮬레이션 단계인 1차 시뮬레이션을 수행하면서 후단 시뮬레이션 단계인 1차 시뮬레이션 이후에 수행되는 1 회 이상의 시뮬레이션들을 시뮬레이션 구간들이나 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 1 이상의 설계블록들에 대하여 시뮬레이션들이 이루어질 수 있도록 하는데 필요한 최소한의 정보를 상기 앞단 시뮬레이션 단계인 1차 시뮬레이션 과정에서 자동적으로 수집할 수 있도록 하고, 상기 1 이상의 시뮬레이터를 이용한 상기 앞단 시뮬레이션 단계인 1차 시뮬레이션을 상기 최소한의 정보를 수집하면서 빠르게 수행하고, 이 수집된 정보를 이용하여 상기 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션들을 가시도를 얻으면서도 신속하게 수행함으로써 시뮬레이션의 속도와 디버깅을 위한 가시도를 동시에 확보하는 것을 가능하게 하는 설계 검증 장치.

### 【청구항 2】

시뮬레이션을 여러 개의 테스트벤치로써 수 차례 수행하여 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 1 이상의 설계 오류들의 위치를

알아내고 이를 수정하는 설계 검증 방법에 있어서,

상기 수 차례의 시뮬레이션들에서의 각각의 시뮬레이션 과정 하나 하나를 앞단 시뮬레이션 단계인 1차 시뮬레이션과 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션들로 나누어서 수행하며, 상기 앞단 시뮬레이션 단계인 1차 시뮬레이션을 수행하면서 상기 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션들을 시뮬레이션 구간들이나 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 1 이상의 설계블록들에 대하여 시뮬레이션들이 이루어질 수 있도록 하는데 필요한 최소한의 정보를 상기 앞단 시뮬레이션 단계인 1차 시뮬레이션 과정에서 자동적으로 수집할 수 있도록 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 부가 코드나 부가 회로를 검증 소프트웨어를 이용하여 자동화된 방식으로 추가하고, 상기 앞단 시뮬레이션 단계인 1차 시뮬레이션을 상기 최소한의 정보를 수집하면서 빠르게 수행하고, 이 수집된 정보를 이용하여 상기 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션들을 가시도를 얻으면서도 신속하게 수행함으로써 시뮬레이션의 속도와 디버깅을 위한 가시도를 동시에 확보하는 것을 가능하게 하는 검증 방법.

### **【청구항 3】**

제 1 항 내지는 제 2 항 내지는 제 12 항 내지는 제 54 항 내지는 제 55 항에 있어서,

최소한의 정보 수집이 앞단 시뮬레이션 단계인 1차 시뮬레이션 과정에서 일정 간격으로 혹은 1 이상의 특정 시뮬레이션 시점들에서 시뮬레이션의 상태를 1 이



상의 파일 형태로 저장하는 것을 포함하는 설계 검증 방법

#### 【청구항 4】

제 1 항 내지는 제 2 항 내지는 제 12 항 내지는 제 54 항 내지는 제 55 항에 있어서,

최소한의 정보 수집이 앞단 시뮬레이션 단계인 1차 시뮬레이션 과정에서 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 1 이상의 설계블록들의 입력과 입출력을 시뮬레이션 전체 과정에 걸쳐서 지속적으로 탐침하여 1 이상의 파일 형태로 저장하여 이를 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션 과정에서 입력스티뮬러스로 이용하는 것을 포함하는 설계 검증 방법.

#### 【청구항 5】

제 3 항에 있어서,

앞단 시뮬레이션 단계인 1차 시뮬레이션 시행 중에 1 이상의 시뮬레이션 시점에서 저장된 1 이상의 시뮬레이션 상태들 중에서 1 이상의 시뮬레이션 상태를 선정하여 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션들을 상기 선정된 1 이상의 시뮬레이션 상태에서부터 시작될 수 있도록 시뮬레이터를 1회 이상 설정한 후에 설정된 시뮬레이션 상태로부터 시뮬레이션을 1회 이상 진행하면서 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 시그널들과 변수들에 대한 탐침을 수행하는 설계 검증 방법

## 【청구항 6】

제 5 항에 있어서,

후단 시뮬레이션 단계인 1차 이후의 시뮬레이션들을 상기 선정된 2 이상의 시뮬레이션 상태들에서부터 시작될 수 있도록 하나의 시뮬레이터를 2회 이상 순서적으로 설정하고 2회 이상 순서적으로 시뮬레이션 진행하는 과정에 있어서, 앞단 시뮬레이션 단계인 1차 시뮬레이션 수행 시에 시뮬레이션의 상태를 저장하는 시뮬레이션 시점이 시뮬레이션 시간적으로 제일 뒤에 있는 시뮬레이션 상태를 우선 상기 하나의 시뮬레이터에 설정하여 이 시뮬레이션의 상태를 가지고 시뮬레이션을 진행하고, 이 후의 시뮬레이션들도 상기 단일 시뮬레이터로써 수행하는 과정에서 상태를 저장한 시뮬레이션 시점이 시뮬레이션 시간적으로 뒤에 있는 것들을 우선으로 하여 시뮬레이터에 설정하여 시뮬레이션들이 시뮬레이션 시간적으로 제일 뒤에서부터 앞서는 순서대로 진행하면서 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 시그널들과 변수들에 대한 탐침을 수행하는 설계 검증 방법

## 【청구항 7】

제 1 항 내지는 제 2 항 내지는 제 3 항 내지는 제 5 항 내지는 제 6 항 내지는 제 12 항 내지는 제 54 항 내지는 제 55 항에 있어서,

앞단 시뮬레이션 단계인 1차 시뮬레이션 시행 중에 1 이상의 시뮬레이션 시점에서 시뮬레이션 상태를 저장하는 방법을 시뮬레이터의 save 명령어 혹은 checkpoint 명령어를 사용하고, 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시

플레이션들을 상기 선정된 1 이상의 시뮬레이션 상태에서부터 시작될 수 있도록 시뮬레이터를 1회 이상 설정한 후에 시뮬레이션을 진행하는 방법을 시뮬레이터의 restart 명령어 혹은 restore 명령어를 사용하는 설계 검증 방법

#### **【청구항 8】**

제 4 항에 있어서,

앞단 시뮬레이션 단계인 1차 시뮬레이션 실행 중에 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 1 이상의 설계블록들에 대한 입력과 입출력들을 시뮬레이션 전 과정에서 탐침하여 저장한 1 이상의 탐침 파일들 중에서 1 이상을 선정하여 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션들을 상기 선정된 1 이상의 탐침 파일과 해당 1 이상의 설계블록들을 이용하여 시뮬레이션 컴파일하여 생성된 1 이상의 시뮬레이션 실행 파일들 중에서 추가적인 탐침이 필요한 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 시그널들과 변수들을 가지고 있는 해당 1 이상의 설계블록들을 컴파일된 형태로 가지고 있는 1 이상의 시뮬레이션 실행 파일을 실행시키면서 상기 추가적인 탐침을 수행함으로써 빠른 시뮬레이션 속도와 설계블록들에 대한 가시도를 신속하게 확보하는 설계 검증 방법

#### **【청구항 9】**

제 1 항 내지는 제 2 항 내지는 제 3 항 내지는 제 5 항 내지는 제 6 항 내지는 제 7 항 내지는 제 12 항 내지는 제 54 항 내지는 제 55 항에 있어서,

후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션을 앞단 시뮬레이

선 단계인 1차 시뮬레이션 과정 중에서 저장된 2 이상의 시뮬레이션 상태들 각각을 이용하여 2회 이상의 부분 시뮬레이션을 하는 것으로 전환하여, 컴퓨터 네트워크로 연결된 2 이상의 컴퓨터에 인스톨된 2 이상의 시뮬레이터들을 이용하여 상기 2 이상의 시뮬레이터 상태들 각각을 상기 2 이상의 시뮬레이터 각각에 설정하여 2 이상의 시뮬레이션들이 서로 독립적으로 상기 2 이상의 시뮬레이터로써 병렬적으로 실행하여 빠른 시뮬레이션 속도를 제공하는 설계 검증 방법

### 【청구항 10】

제 1 항 내지는 제 2 항 내지는 제 4 항 내지는 제 8 항 내지는 제 12 항 내지는 제 54 항 내지는 제 55 항에 있어서,

후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션을 앞단 시뮬레이션 단계인 1차 시뮬레이션을 통하여 얻어진 2 이상의 설계블록들과 이들 설계블록들 각각의 입력과 입출력을 앞단 시뮬레이션 단계인 1차 시뮬레이션 전과정에서 탐침한 2 이상의 탐침 파일들을 시뮬레이션 컴파일하여 얻어진 2 이상의 시뮬레이션 실행 파일들을 컴퓨터 네트워크로 연결된 2 이상의 컴퓨터에 인스톨된 2 이상의 시뮬레이터들을 이용하여 독립적으로 병렬적으로 실행하여 빠른 시뮬레이션 속도를 제공하는 설계 검증 방법

### 【청구항 11】

제 1 항 내지는 제 2 항 내지는 제 4 항 내지는 제 8 항 내지는 제 12 항 내지는 제 54 항 내지는 제 55 항에 있어서,

후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션을 앞단 시뮬레이션 단계인 1차 시뮬레이션을 통하여 얻어진 2 이상의 설계블록들과 이들 설계블록들 각각의 입력과 입출력을 앞단 시뮬레이션 단계인 1차 시뮬레이션 전과정에서 탐침한 2 이상의 탐침 파일들을 변환한 2 이상의 테스트벤치 파일들을 시뮬레이션 컴파일하여 얻어진 2 이상의 시뮬레이션 실행 파일들을 컴퓨터 네트워크로 연결된 2 이상의 컴퓨터에 인스톨된 2 이상의 시뮬레이터들을 이용하여 독립적으로 병렬적으로 실행하여 빠른 시뮬레이션 속도를 제공하는 설계 검증 방법

### 【청구항 12】

임의의 시뮬레이션 수행 내지는 시뮬레이션가속 수행 내지는 하드웨어에물레이션 수행 내지는 프로토타이핑 수행을 통하여 얻어진 결과를 이용한 추가적인 시뮬레이션 수행에서,

상기 추가적인 시뮬레이션 수행을 앞단 시뮬레이션 단계인 1차 1회의 시뮬레이션과 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션들로 나누어서 수행하며, 상기 1차 시뮬레이션을 수행하면서 상기 1차 이후의 1회 이상의 시뮬레이션을 시뮬레이션 구간들이나 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 1 이상의 설계블록들에 한정하여 시뮬레이션들이 이루어질 수 있도록 하는데 필요한 최소한의 정보를 1차 시뮬레이션 과정에서 자동적으로 수집할 수 있도록 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 부가 코드나 부가 회로를 검증 소프트웨어를 이용하여 자동화된 방식으로 추가하고, 상기 1차 시뮬레이션을 수행하면서 상기 최소한의 정보를 수집하고, 이 수집된 정보를 이용

하여 상기 1차 이후의 1회 이상의 시뮬레이션을 신속하게 수행하는 것을 가능하게 하는 검증 방법.

### 【청구항 13】

제 1 항 내지는 제 2 항 내지는 제 3 항 내지는 제 4 항 내지는 제 5 항 내지는 제 9 항 내지는 제 10 항 내지는 제 11 항 내지는 제 12 항 내지는 제 23 항 내지는 제 24 항 내지는 제 25 항 내지는 제 27 항 내지는 제 54 항 내지는 제 55 항에 있어서,

앞단 시뮬레이션 단계인 1차의 시뮬레이션을 병렬 시뮬레이션으로 수행하는 검증 방법

### 【청구항 14】

제 31 항 내지는 제 32 항에 있어서,

앞단 시뮬레이션 단계인 1차의 시뮬레이션을 사이클-기반 시뮬레이션으로 수행하고, 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션은 이벤트-구동 시뮬레이션으로 수행하는 검증 방법

앞단 시뮬레이션 단계인 1차의 시뮬레이션을 DUV의 전체 내지는 DUV내의 1 이상의 설계객체들에 대하여서 사이클-기반 시뮬레이션으로 수행하고, 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션은 이벤트-구동 시뮬레이션으로 수행하는 검증 방법

### 【청구항 15】

제 31 항 내지는 제 32 항에 있어서,

앞단 시뮬레이션 단계인 1차의 시뮬레이션을 DUV의 전체 내지는 DUV내의 1 이상의 설계객체들에 대하여서 SystemC 시뮬레이터를 이용하여 수행하고, 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션은 Verilog 시뮬레이터 내지는 VHDL 시뮬레이터 내지는 SystemVerilog 시뮬레이터를 이용하여 수행하는 검증 방법

### 【청구항 16】

제 31 항 내지는 제 32 항에 있어서,

앞단 시뮬레이션 단계인 1차의 시뮬레이션을 DUV의 전체 내지는 DUV내의 1 이상의 설계객체들에 대하여서 RTL 시뮬레이션으로 수행하고, 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션은 게이트수준 시뮬레이션으로 수행하는 검증 방법

### 【청구항 17】

제 14 항 내지는 제 15 항 내지는 제 16 항에 있어서,

상기 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션을 2 이상의 시뮬레이터들을 이용하여서 병렬적으로 진행하는 검증 방법

### 【청구항 18】

시뮬레이션을 여러 개의 테스트벤치로써 수 차례 수행하여 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 1 이상의 설계 오류들의 위치를

알아내고 이를 수정하는 설계 검증 방법에 있어서,

앞단 시뮬레이션 단계인 설계 검증의 1차 시뮬레이션 과정에서 2 이상의 시뮬레이션 시점들에서 2 이상의 시뮬레이션 상태를 저장하고, 후단 시뮬레이션 단계인 1차 이후에 처음으로 수행되는 시뮬레이션에서 2 이상의 시뮬레이터를 이용하여 상기 1차 시뮬레이션 과정에서 저장된 2 이상의 시뮬레이션 상태로부터 시뮬레이션이 병렬적으로 진행할 수 있도록 하고, 이 병렬적 시뮬레이션이 진행되는 과정에서 설계 코드에 존재하는 1 이상의 특정 설계블록들의 입력과 입출력값들을 병렬적으로 탐침하여 얻어진 2 이상의 덤프 파일들이나 이들을 테스트벤치로 변환된 파일들을 해당 설계블록들과 같이 2 이상의 시뮬레이터들을 이용하여 동시에 시뮬레이션 컴파일하여 2 이상의 시뮬레이션 실행파일들을 생성하고, 상기 2 이상의 컴퓨터들에서 병렬적으로 시뮬레이션 하는 과정을 통하여 빠른 시뮬레이션 속도와 설계블록들에 대한 가시도를 확보하는 설계 검증 방법

#### **【청구항 19】**

제 14 항 내지는 제 16 항에 있어서,

상기 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션을 게이트수준에서 이벤트-구동 시뮬레이션으로 수행하는 검증 방법

#### **【청구항 20】**

제 19 항에 있어서,

상기 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션을 게이트수



준에서 이벤트-구동 시뮬레이션을 배치 및 배선 결과에서 추출된 타이밍정보를 활용한 타이밍 시뮬레이션으로 수행하는 검증 방법

#### **【청구항 21】**

제 14 항 내지는 제 15 항 내지는 제 19 항 내지는 제 20 항에 있어서,

앞단 시뮬레이션 단계인 1차의 시뮬레이션 과정에서 1 이상의 시뮬레이션 시점에서 가시도가 필요한 DUV에서의 1 이상의 설계객체들에 대한 상태정보를 얻고, 이렇게 얻어진 1 이상의 설계검증대상의 상태정보를 이용하여 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션을 수행하는 검증 방법

#### **【청구항 22】**

제 14 항 내지는 제 15 항 내지는 제 19 항 내지는 제 20 항에 있어서,

앞단 시뮬레이션 단계인 1차의 시뮬레이션 과정에서 2 이상의 시뮬레이션 시점에서 가시도가 필요한 DUV에서의 1 이상의 설계객체들에 대한 상태정보를 얻고, 이렇게 얻어진 2 이상의 설계검증대상의 상태정보를 이용하여 후단 시뮬레이션 단계인 1차 이후의 2회 이상의 시뮬레이션을 병렬적으로 수행하는 검증 방법

#### **【청구항 23】**

제 1 항 내지는 제 2 항 내지는 제 12 항 내지는 제 54 항 내지는 제 55 항에 있어서,

최소한의 정보 수집이 앞단 시뮬레이션 단계인 1차 시뮬레이션 과정에서 일정 간격으로 혹은 1 이상의 특정 시뮬레이션 시점들에서 가시도가 필요한 DUV에서

의 1 이상의 설계객체들에 대한 상태저장과 더불어, 시뮬레이션 전 구간에 걸쳐서 상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 모든 입력 값들과 입력모드시의 모든 입출력 값들을 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션 과정에서 입력스티뮬러스로 이용하기 위하여 상기 입력들에서나 입출력들에서 매 이벤트 발생시 내지는 매 사이클마다 내지는 매 트랜잭션마다의 저장을 1 이상의 파일형태로 저장하거나 혹은 상기 일정 간격으로 혹은 1 이상의 특정 시뮬레이션 시점들에서 TB의 상태저장을 1 이상의 파일 형태로 저장하는 것을 포함하는 설계 검증 방법

#### **【청구항 24】**

제 1 항 내지는 제 2 항 내지는 제 12 항 내지는 제 54 항 내지는 제 55 항에 있어서,

최소한의 정보 수집이 앞단 시뮬레이션 단계인 1차 시뮬레이션 과정에서 일정 간격으로 혹은 1 이상의 특정 시뮬레이션 시점들에서 가시도가 필요한 DUV에서의 1 이상의 설계객체들에 대한 상태와 상기 앞단 시뮬레이션 단계인 1차 시뮬레이션 전체 과정에서 상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 모든 입력 값들과 입력모드시의 모든 입출력 값들을 후단 시뮬레이션 단계인 1차 이후의 시뮬레이션 과정에서 입력스티뮬러스로 이용하기 위하여 상기 입력들에서나 입출력들에서 매 이벤트 발생시 내지는 매 사이클마다 내지는 매 트랜잭션마다의 저장을 1 이상의 파일 형태로 저장하는 것을 포함하는 설계 검증 방법

## 【청구항 25】

제 23 항 내지는 제 24 항에 있어서,

앞단 시뮬레이션 단계인 1차 시뮬레이션 시행 중에 1 이상의 시뮬레이션 시점에서 저장된 1 이상의 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태와 경우에 따라서는 TB의 상태들 중에서 1 이상의 상태를 선정하여 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션들을 상기 선정된 1 이상의 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태와 경우에 따라서는 TB의 상태에서부터 시작될 수 있도록 시뮬레이터를 1회 이상 설정한 후에 시뮬레이션을 1회 이상 진행하면서 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 시그널들과 변수들에 대한 탐침을 수행함으로써 빠른 시뮬레이션 속도와 설계블럭들에 대한 가시도를 신속하게 확보하는 설계 검증 방법

## 【청구항 26】

제 25 항에 있어서,

후단 시뮬레이션 단계인 1차 이후의 시뮬레이션들을 상기 선정된 2 이상의 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태와 경우에 따라서는 TB의 상태들에서부터 시작될 수 있도록 하나의 시뮬레이터를 2회 이상 순서적으로 설정하고 2회 이상 순서적으로 시뮬레이션 진행하는 과정에 있어서, 앞단 시뮬레이션 단계인 1차 시뮬레이션 수행 시에 상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태와 경우에 따라서는 TB의 상태를 저장하는 시뮬레이션 시점이 시뮬레

이전 시간적으로 제일 뒤에 있는 DUV에서의 1 이상의 설계객체들의 상태와 경우에 따라서는 TB의 상태를 우선 상기 하나의 시뮬레이터에 설정하여 이 상태를 가지고 시뮬레이션을 진행하고, 이 후의 시뮬레이션들도 상기 단일 시뮬레이터로써 수행하는 과정에서 상태를 저장한 시뮬레이션 시점이 시뮬레이션 시간적으로 뒤에 있는 것들을 우선으로 하여 시뮬레이터에 설정하여 시뮬레이션들이 시뮬레이션 시간적으로 제일 뒤에서부터 앞서는 순서대로 진행하면서 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 시그널들과 변수들에 대한 탐침을 수행함으로써 빠른 시뮬레이션 속도와 설계블럭들에 대한 가시도를 신속하게 확보하는 설계 검증 방법

#### **【청구항 27】**

제 1 항 내지는 제 2 항 내지는 제 23 항 내지는 제 24 항 내지는 25 항 내지는 제 26 항 내지는 제 54 항 내지는 제 55 항 에 있어서,

후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션을 앞단 시뮬레이션 단계인 1차 시뮬레이션 과정 중에서 저장된 2 이상의 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태와 경우에 따라서는 TB의 상태들 각각을 이용하여 2회 이상의 부분 시뮬레이션을 하는 것으로 전환하여, 컴퓨터 네트워크로 연결된 2 이상의 컴퓨터에 인스톨된 2 이상의 시뮬레이터들을 이용하여 상기 2 이상의 DUV에서의 1 이상의 설계객체들의 상태와 경우에 따라서는 TB의 상태들 각각을 상기 2 이상의 시뮬레이터 각각에 설정하여 2 이상의 시뮬레이션들이 서로 독립적으로 상기 2 이상의 시뮬레이터로써 병렬적으로 실행하여 빠른 시뮬레이션 속도와 설계블

력들에 대한 가시도를 신속하게 확보하는 설계 검증 방법

#### 【청구항 28】

제 1 항 내지는 제 2 항 내지는 제 3 항 내지는 제 4 항 내지는 제 5 항 내지는 제 9 항 내지는 제 10 항 내지는 제 11 항 내지는 제 12 항 내지는 제 23 항 내지는 제 24 항 내지는 제 25 항 내지는 제 27 항 내지는 제 54 항 내지는 제 55 항에 있어서,

앞단 시뮬레이션 단계인 1차의 시뮬레이션을 빠른 시뮬레이션이 수행될 수 있도록 원래의 설계 코드 대신에 원래 설계 코드와 함수적으로 등가이나 다른 구문 형식으로 변환된 설계 코드를 사용하여 시뮬레이션하는 검증 방법

#### 【청구항 29】

시뮬레이션을 여러 개의 테스트벤치로써 수 차례 수행하여 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 1 이상의 설계 오류들의 위치를 알아내고 이를 수정하는 설계 검증 방법에 있어서,

설계 검증의 앞단 시뮬레이션 단계인 1차 시뮬레이션 과정에서 2 이상의 시뮬레이션 시점들에서 2 이상의 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태와 경우에 따라서는 TB의 상태들을 저장하고, 후단 시뮬레이션 단계인 1차 이후에 처음으로 수행되는 시뮬레이션에서 2 이상의 시뮬레이터를 이용하여 상기 1차 시뮬레이션 과정에서 저장된 2 이상의 상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태와 경우에 따라서는 TB의 상태들로부터 시뮬레이션이 병렬적으로

진행할 수 있도록 하고, 이 병렬적 시뮬레이션이 진행되는 과정에서 설계 코드에 존재하는 1 이상의 특정 설계블록들의 입력과 입출력값들을 병렬적으로 탐침하여 얻어진 2 이상의 덤프 파일들이나 이들을 테스트벤치로 변환된 파일들을 해당 설계 블록들과 같이 2 이상의 시뮬레이터들을 이용하여 동시에 시뮬레이션 컴파일하여 2 이상의 시뮬레이션 실행파일들을 생성하고, 상기 2 이상의 컴퓨터들에서 병렬적으로 시뮬레이션 하는 과정을 통하여 빠른 시뮬레이션 속도와 설계블럭들에 대한 가시도를 확보하는 설계 검증 방법

### 【청구항 30】

제 21 항 내지는 제 22 항 내지는 제 23 항 내지는 제 24 항 내지는 제 25 항 내지는 제 26 항 내지는 제 27 항 내지는 제 29 항에 있어서,

상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태 정보로 상기 1 이상의 해당 설계객체들의 최소상태 정보를 사용하는 검증 방법

### 【청구항 31】

제 1 항 내지는 제 2 항 내지는 제 3 항 내지는 제 4 항 내지는 제 5 항 내지는 제 9 항 내지는 제 10 항 내지는 제 11 항 내지는 제 12 항 내지는 제 23 항 내지는 제 24 항 내지는 제 25 항 내지는 제 27 항 내지는 제 54 항 내지는 제 55 항에 있어서,

앞단 시뮬레이션 단계인 1차 시뮬레이션에서 최소한 1 이상의 설계객체들에 대해서 후단 시뮬레이션 단계인 1차 이후의 1 이상의 시뮬레이션들보다 더 추상화

된 설계 코드를 이용함으로써 더 추상화된 수준에서 상기 앞단 시뮬레이션 단계인 1차 시뮬레이션을 수행하고, 상기 후단 시뮬레이션 단계인 1차 이후의 1 이상의 시뮬레이션은 상기 앞단 시뮬레이션 단계인 1차 시뮬레이션보다 덜 추상화된 설계 코드를 이용하여 시뮬레이션을 수행함으로써 신속한 검증을 수행하는 검증 방법

### 【청구항 32】

제 31 항에 있어서,

상기 앞단 시뮬레이션 단계인 1차 시뮬레이션의 결과로 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태 정보를 1 이상의 시뮬레이션 시점에서 저장하고, 후단 시뮬레이션 단계인 1차 이후의 1 이상의 시뮬레이션에서 상기 앞단 시뮬레이션 단계인 1차 시뮬레이션에서 저장된 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 1 이상의 상태 정보를 이용하여서 1회 이상의 시뮬레이션을 2 이상의 시뮬레이터를 사용하여 병렬적으로 수행하거나 단일 시뮬레이터를 이용하여서 순차적으로 수행함으로써 신속한 검증을 수행하는 검증 방법

### 【청구항 33】

제 23 항 내지는 제 24 항에 있어서,

앞단 시뮬레이션 단계인 1차 시뮬레이션 시행 중에 1 이상의 시뮬레이션 시점에서 저장된 1 이상의 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태와 상기 입력스티뮬러스로 이용하기 위하여 저장된 1 이상의 파일을 선정하여 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션들을 상기 선정된 1 이상의 가

시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태에서부터 시작될 수 있도록 시뮬레이터를 1회 이상 설정한 후에 시뮬레이션을 1회 이상 진행하면서 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 시그널들과 변수들에 대한 탐침을 수행함으로써 빠른 시뮬레이션 속도와 설계블럭들에 대한 가시도를 신속하게 확보하는 설계 검증 방법

### 【청구항 34】

제 33 항에 있어서,

후단 시뮬레이션 단계인 1차 이후의 시뮬레이션들을 상기 선정된 2 이상의 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태들에서부터 시작될 수 있도록 하나의 시뮬레이터를 2회 이상 순서적으로 설정하고 2회 이상 순서적으로 시뮬레이션 진행하는 과정에 있어서, 앞단 시뮬레이션 단계인 1차 시뮬레이션 수행 시에 상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태를 저장하는 시뮬레이션 시점이 시뮬레이션 시간적으로 제일 뒤에 있는 DUV에서의 1 이상의 설계객체들의 상태를 우선 상기 하나의 시뮬레이터에 설정하여 이 상태를 가지고 시뮬레이션을 진행하고, 이 후의 시뮬레이션들도 상기 단일 시뮬레이터로써 수행하는 과정에서 상태를 저장한 시뮬레이션 시점이 시뮬레이션 시간적으로 뒤에 있는 것들을 우선으로 하여 시뮬레이터에 설정하여 시뮬레이션들이 시뮬레이션 시간적으로 제일 뒤에서부터 앞서는 순서대로 진행하면서 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 시그널들과 변수들에 대한 탐침을 수행함으로써 빠른 시뮬레이션 속도와 설계블럭들에 대한 가시도를 신속하게 확보하는 설계 검증 방법



### 【청구항 35】

제 1 항 내지는 제 2 항 내지는 제 23 항 내지는 제 24 항 내지는 제 33 항 내지는 제 34 항 내지는 제 54 항 내지는 제 55 항에 있어서,

후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션을 앞단 시뮬레이션 단계인 1차 시뮬레이션 과정 중에서 저장된 2 이상의 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태들 각각을 이용하여 2회 이상의 부분 시뮬레이션을 하는 것으로 전환하여, 컴퓨터 네트워크로 연결된 2 이상의 컴퓨터에 인스톨된 2 이상의 시뮬레이터들을 이용하여 상기 2 이상의 DUV에서의 1 이상의 설계객체들의 상태들 각각을 상기 2 이상의 시뮬레이터 각각에 설정하여 2 이상의 시뮬레이션들이 서로 독립적으로 상기 2 이상의 시뮬레이터로써 병렬적으로 실행하여 빠른 시뮬레이션 속도와 설계블럭들에 대한 가시도를 신속하게 확보하는 설계 검증 방법

### 【청구항 36】

시뮬레이션을 여러 개의 테스트벤치로써 수 차례 수행하여 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 1 이상의 설계 오류들의 위치를 알아내고 이를 수정하는 설계 검증 방법에 있어서,

앞단 시뮬레이션 단계인 설계 검증의 1차 시뮬레이션 과정에서 2 이상의 시뮬레이션 시점들에서 2 이상의 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태와 시뮬레이션 전 구간에 걸쳐서 상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 모든 입력 값들과 입력모드시의 모든 입출력 값들을 이 입력들에서나

입출력들에서 매 이벤트 발생시마다 내지는 매 사이클마다 내지는 매 트랜잭션 마다 저장하고, 후단 시뮬레이션 단계인 1차 이후에 처음으로 수행되는 시뮬레이션에서 2 이상의 시뮬레이터를 이용하여 상기 1차 시뮬레이션 과정에서 저장된 2 이상의 상기 가시도가 필요한 DUV에서의 1 이상의 설계객체들의 상태들로부터 시뮬레이션이 병렬적으로 진행할 수 있도록 하고, 이 병렬적 시뮬레이션이 진행되는 과정에서 설계 코드에 존재하는 1 이상의 특정 설계블록들의 입력과 입출력값들을 병렬적으로 탐침하여 얻어진 2 이상의 덤프 파일들이나 이들을 테스트벤치로 변환된 파일들을 해당 설계블록들과 같이 2 이상의 시뮬레이터들을 이용하여 동시에 시뮬레이션 컴파일하여 2 이상의 시뮬레이션 실행파일들을 생성하고, 상기 2 이상의 컴퓨터들에서 병렬적으로 시뮬레이션 하는 과정을 통하여 빠른 시뮬레이션 속도와 설계블록들에 대한 가시도를 확보하는 설계 검증 방법

### **【청구항 37】**

상위수준에서의 시뮬레이션 결과를 이용하여서 하위수준에서의 시뮬레이션을 신속하게 수행하는 검증 방법

### **【청구항 38】**

제 37 항 내지는 제 57 항 내지는 제 58 항에 있어서,

상기 상위수준에서의 시뮬레이션 결과가 1 이상의 설계객체들에 대한 상태 정보를 포함하는 검증 방법

**【청구항 39】**

제 38 항 내지는 제 64 항 내지는 제 65 항에 있어서,

상기 1 이상의 설계객체들에 대한 상태정보가 완전상태 정보인 검증 방법

**【청구항 40】**

제 38 항 내지는 제 64 항 내지는 제 65 항에 있어서,

상기 1 이상의 설계객체들에 대한 상태정보가 최소상태 정보인 검증 방법

**【청구항 41】**

제 38 항 내지는 제 39 항 내지는 제 40 항 내지는 제 64 항 내지는 제 65 항에 있어서,

하위수준에서의 시뮬레이션을 신속하게 수행하는 방법이 상기 1 이상의 설계 객체들에 대한 상기 상태 정보를 이용하여서 하위수준에서 2 이상의 시뮬레이션들을 병렬적으로 수행하는 검증 방법

**【청구항 42】**

제 41 항에 있어서,

병렬적 수행이 시간적 병렬 수행인 검증 방법

**【청구항 43】**

제 41 항에 있어서,

병렬적 수행이 공간적 병렬 수행인 검증 방법

**【청구항 44】**

제 37 항에 있어서,

상위수준에서의 시뮬레이션 결과와 하위수준에서의 시뮬레이션 결과 중 하나를 레퍼런스로 가정하고, 1 이상의 특정 시뮬레이션 시점에서 하위수준에서의 시뮬레이션 결과가 상위수준에서의 시뮬레이션 결과를 비교하여서, 하위수준에서의 시뮬레이션 결과와 상위수준에서의 시뮬레이션 결과가 다른 상황을 찾아내어 설계를 수정함으로서, 상기 1 이상의 특정 시뮬레이션 시점에서 하위수준에서의 시뮬레이션 결과와 상위수준에서의 시뮬레이션 결과를 같게 만드는 설계 검증 및 설계 수정 방법

#### **【청구항 45】**

제 44 항 내지는 제 59 항에 있어서,

상기 상위수준에서의 시뮬레이션이 사이클-기반이고, 상기 하위수준에서의 시뮬레이션이 이벤트-구동인 설계 검증 및 설계 수정 방법

#### **【청구항 46】**

제 44 항 내지는 제 59 항에 있어서,

상기 상위수준에서의 시뮬레이션이 SystemC 시뮬레이션이고, 상기 하위수준에서의 시뮬레이션이 RTL 시뮬레이션인 설계 검증 및 설계 수정 방법

#### **【청구항 47】**

제 44 항 내지는 제 59 항에 있어서,

상기 상위수준에서의 시뮬레이션이 RTL 시뮬레이션이고, 상기 하위수준에서

의 시뮬레이션이 게이트수준 시뮬레이션인 설계 검증 및 설계 수정 방법

**【청구항 48】**

제 47 항에 있어서,

상기 게이트수준 시뮬레이션이 타이밍 정보를 이용한 타이밍 시뮬레이션인  
설계 검증 및 설계 수정 방법

**【청구항 49】**

제 44 항 내지는 제 59 항에 있어서,

상기 하위수준에서의 시뮬레이션 결과가 상위수준에서의 시뮬레이션 결과와  
다른 상황을 신속하게 찾아내기 위하여서, 상위수준에서의 시뮬레이션 결과를 이용  
하여 하위수준에서 시뮬레이션을 수행하는 설계 검증 및 설계 수정 방법

**【청구항 50】**

제 44 항 내지는 제 59 항에 있어서,

상기 하위수준에서의 시뮬레이션 결과가 상위수준에서의 시뮬레이션 결과와  
다른 상황을 신속하게 찾아내기 위하여서, 상위수준에서의 시뮬레이션 결과를 이용  
하여 하위수준에서 병렬적으로 시뮬레이션을 수행하는 설계 검증 및 설계 수정 방  
법

**【청구항 51】**

제 49 항 내지는 제 50 항에 있어서,

상기 하위수준에서의 시뮬레이션 결과가 상위수준에서의 시뮬레이션 결과와

다른 상황을 신속하게 찾아내기 위하여서, 상기 하위수준에서 이용하는 상위수준에서의 시뮬레이션 결과가 1 이상의 설계객체들에 대한 상태 정보를 포함하는 설계 검증 및 설계 수정 방법

#### **【청구항 52】**

제 51 항에 있어서,

상기 1 이상의 설계객체들에 대한 상태정보가 상기 1 이상의 설계객체들에 대한 완전상태 정보인 설계 검증 및 설계 수정 방법

#### **【청구항 53】**

제 51 항에 있어서,

상기 1 이상의 설계객체들에 대한 상태정보가 상기 1 이상의 설계객체들에 대한 최소상태 정보인 설계 검증 및 설계 수정 방법

#### **【청구항 54】**

검증 소프트웨어와 1 이상의 시뮬레이터를 구비하는 설계검증 장치에 있어서,

상기 검증 소프트웨어는 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 자동화된 방식을 통하여 부가 코드나 부가 회로를 추가하여 앞단 시뮬레이션 단계인 1차 시뮬레이션을 수행하면서 후단 시뮬레이션 단계인 1차 시뮬레이션 이후에 수행되는 1 회 이상의 시뮬레이션들을 시뮬레이션 구간들이나 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 1 이상의 설계블록들에 대하

여 시뮬레이션들이 이루어질 수 있도록 하는데 필요한 최소한의 정보를 상기 앞단 시뮬레이션 단계인 1차 시뮬레이션 과정에서 자동적으로 수집할 수 있도록 하고, 상기 1 이상의 시뮬레이터를 이용한 상기 앞단 시뮬레이션 단계인 1차 시뮬레이션을 상기 최소한의 정보를 수집하면서 빠르게 수행하고, 이 수집된 정보를 이용하여 상기 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션들을 신속하게 수행하는 것을 가능하게 하는 설계 검증 장치.

### 【청구항 55】

시뮬레이션을 여러 개의 테스트벤치로써 수 차례 수행하여 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 1 이상의 설계 오류들의 위치를 알아내고 이를 수정하는 설계 검증 방법에 있어서,

상기 수 차례의 시뮬레이션들에서의 각각의 시뮬레이션 과정 하나 하나를 앞단 시뮬레이션 단계인 1차 시뮬레이션과 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션들로 나누어서 수행하며, 상기 앞단 시뮬레이션 단계인 1차 시뮬레이션을 수행하면서 상기 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션들을 시뮬레이션 구간들이나 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 1 이상의 설계블록들에 대하여 시뮬레이션들이 이루어질 수 있도록 하는데 필요한 최소한의 정보를 상기 앞단 시뮬레이션 단계인 1차 시뮬레이션 과정에서 자동적으로 수집할 수 있도록 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 부가 코드나 부가 회로를 검증 소프트웨어를 이용하여 자동화된 방식으로 추가하고, 상기 앞단 시뮬레이션 단계인 1차 시뮬레이션을 상기 최소한의 정

보를 수집하면서 빠르게 수행하고, 이 수집된 정보를 이용하여 상기 후단 시뮬레이션 단계인 1차 이후의 1회 이상의 시뮬레이션들을 신속하게 수행하는 검증 방법.

#### **【청구항 56】**

모든 설계 객체들이 변경이 없는 상황에서, 같은 추상화 단계에서 수행되는 앞단 시뮬레이션에서는 시뮬레이션 과정에서 최소한도의 시뮬레이션 실행 결과만을 시뮬레이션 결과로 저장하게 하고, 상기 저장된 앞단 시뮬레이션 결과를 이용하여 같은 추상화 단계에서 수행되는 후단 시뮬레이션을 2 이상의 시뮬레이터를 이용하여 병렬적으로 내지는 하나의 시뮬레이터만을 이용하여서 1회 이상 순차적으로 신속하게 수행하는 동시에, 필요시에 후단 시뮬레이션에서는 1 이상의 설계 객체들에 존재하는 1 이상의 변수들 및 시그널들에 대한 덤프를 수행하여 높은 가시도도 동시에 제공하는 검증 방법

#### **【청구항 57】**

제 37 항에 있어서,

상기 상위수준에서의 시뮬레이션 결과가 시뮬레이션 파형을 저장한 1 이상의 파일을 포함하고, 상기 1 이상의 파일을 하위수준에서 이용하는 존재하는 검증 방법

#### **【청구항 58】**

제 37 항에 있어서,

상기 상위수준에서의 시뮬레이션 결과로 시뮬레이션 파형을 저장한 1 이상의



파일이 존재함으로서, 상기 1 이상의 파일을 하위수준에서 이용하는 존재하는 검증 방법

#### 【청구항 59】

제 37 항에 있어서,

시뮬레이션 파형을 저장한 1 이상의 파일이 상기 상위수준에서의 시뮬레이션 결과로서 존재하고, 이를 하위수준의 시뮬레이션에서 레퍼런스로 활용하여, 1 이상의 특정 시뮬레이션 시점에서 하위수준에서의 시뮬레이션 결과를 상위수준에서의 시뮬레이션 결과와 비교하여서 하위수준에서의 시뮬레이션 결과가 상위수준에서의 시뮬레이션 결과와 다른 상황을 찾아내어 설계를 수정함으로서 상기 1 이상의 특정 시뮬레이션 시점에서 하위수준에서의 시뮬레이션 결과와 상위수준에서의 시뮬레이션 결과가 동일할 수 있도록 하는 설계 검증 및 설계 수정 방법

#### 【청구항 60】

제 44 항 내지는 59 항에 있어서,

상기 1 이상의 특정 시뮬레이션 시점에서 하위수준에서의 시뮬레이션 결과를 상위수준에서의 시뮬레이션 결과와 비교하여서 하위수준에서의 시뮬레이션 결과가 상위수준에서의 시뮬레이션 결과와 다른 상황을 찾아내어 설계를 수정함으로서 상기 1 이상의 특정 시뮬레이션 시점에서 하위수준에서의 시뮬레이션 결과와 상위수준에서의 시뮬레이션 결과가 동일할 수 있도록 하는 것을, 하위수준에서의 시뮬레이션 결과를 상위수준에서의 시뮬레이션 결과와 같아지도록 함으로서 달성하는 설

## 계 검증 및 설계 수정 방법

### 【청구항 61】

제 44 항 내지는 제 59 항에 있어서,

상기 1 이상의 특정 시뮬레이션 시점에서 하위수준에서의 시뮬레이션 결과를 상위수준에서의 시뮬레이션 결과와 비교하여서 하위수준에서의 시뮬레이션 결과가 상위수준에서의 시뮬레이션 결과와 다른 상황을 찾아내어 설계를 수정함으로서 상기 1 이상의 특정 시뮬레이션 시점에서 하위수준에서의 시뮬레이션 결과와 상위수준에서의 시뮬레이션 결과가 동일할 수 있도록 하는 것을, 상위수준에서의 시뮬레이션 결과를 하위수준에서의 시뮬레이션 결과와 같아지도록 함으로서 달성하는 설계 검증 및 설계 수정 방법

### 【청구항 62】

제 44 항 내지는 제 59 항에 있어서,

상기 상위수준에서의 시뮬레이션이 트랜잭션-기반 시뮬레이션이고, 상기 하위수준에서의 시뮬레이션이 사이클-기반 시뮬레이션인 설계 검증 및 설계 수정 방법

### 【청구항 63】

제 44 항 내지는 제 59 항에 있어서,

상기 상위수준에서의 시뮬레이션이 행위수준 시뮬레이션이고, 상기 하위수준에서의 시뮬레이션이 레지스터전송수준 시뮬레이션인 설계 검증 및 설계 수정 방법

#### 【청구항 64】

제 37 항 내지는 제 57 항 내지는 제 58 항에 있어서,

상기 상위수준에서의 시뮬레이션 결과가 1 이상의 설계객체들에 대한 상태 정보와 시뮬레이션 전 구간에 걸친 상기 1 이상의 설계객체들에 대한 모든 입력과 입력모드 시의 입출력 정보를 포함하는 검증 방법

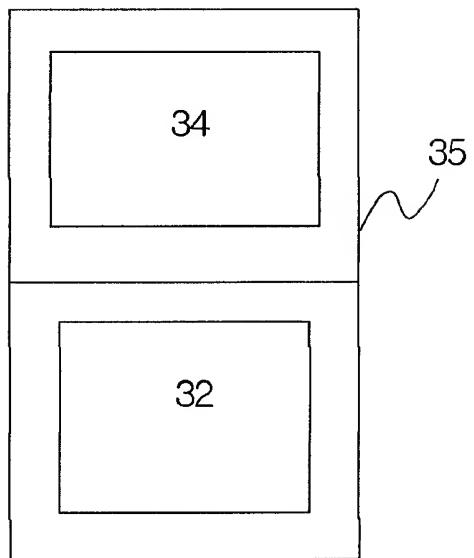
#### 【청구항 65】

제 38 항 내지는 제 64 항에 있어서,

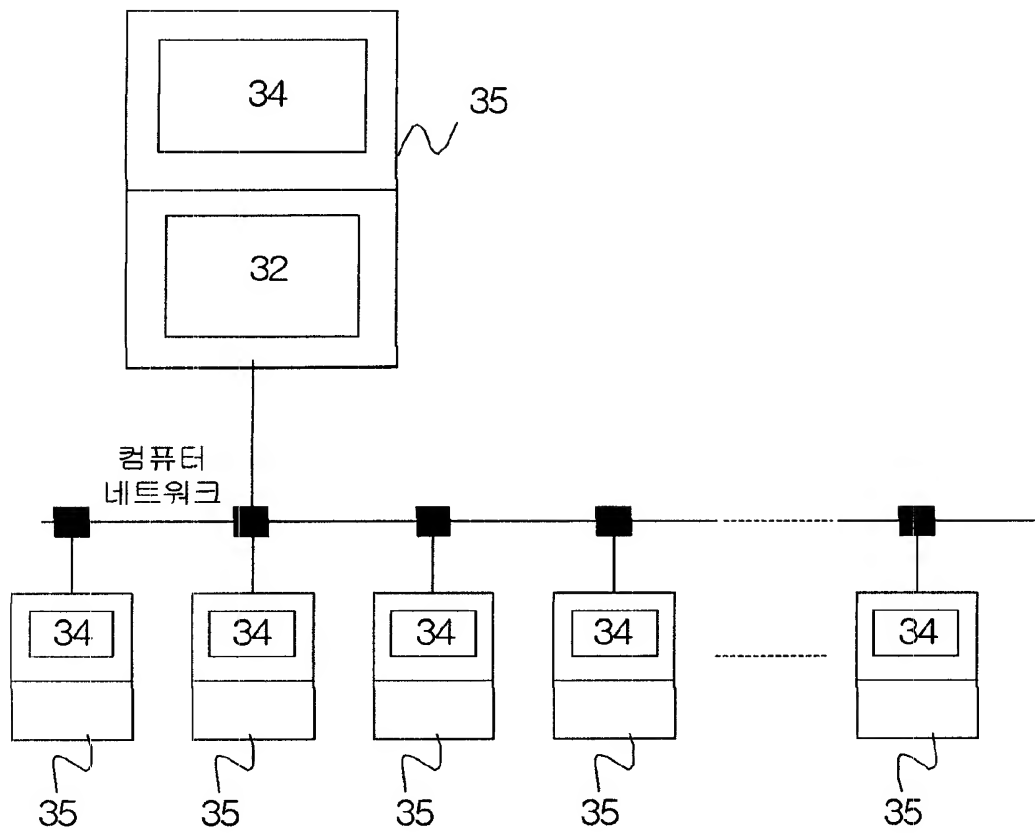
상기 1 이상의 설계객체들에 대한 상태 정보가 시뮬레이션 전 구간 상에서의 모든 상태정보가 아닌 시뮬레이션 구간 상에서의 1 이상의 특정 시뮬레이션 시점들에서만 상태정보인 검증 방법

【도면】

【도 1】

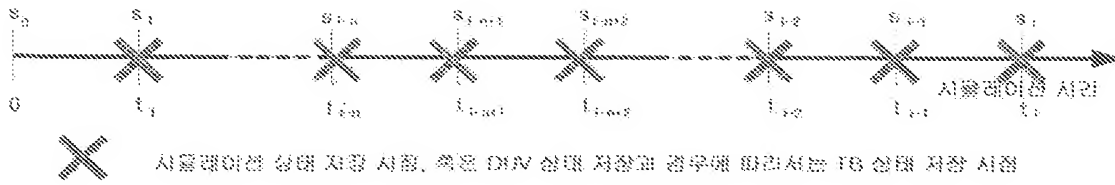


【도 2】

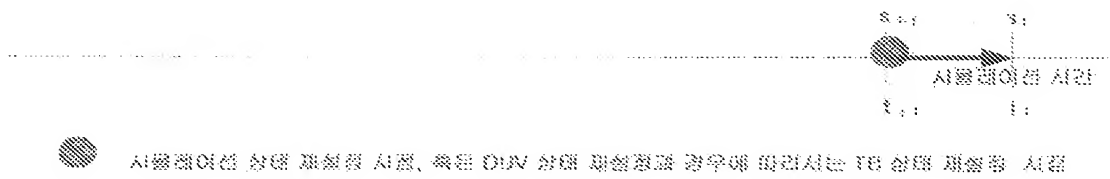


### 【도 3】

#### i) 1차 시뮬레이션

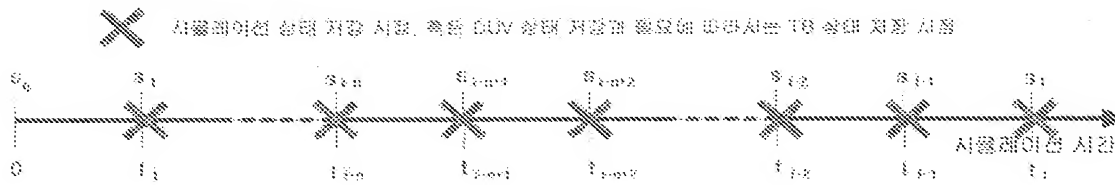


#### ii) 1차 이후의 시뮬레이션

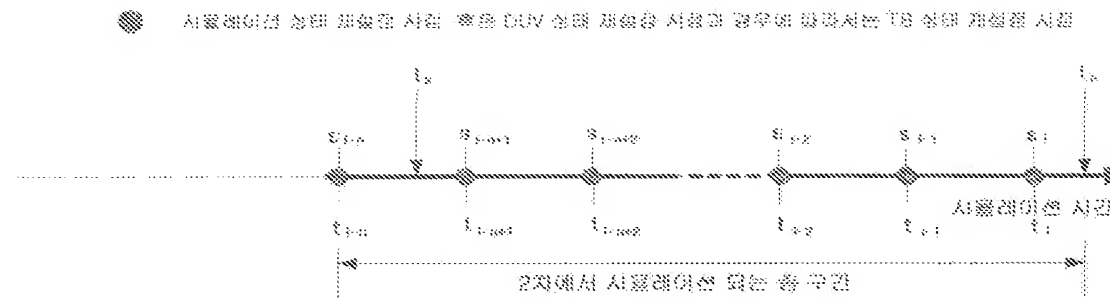


### 【도 4】

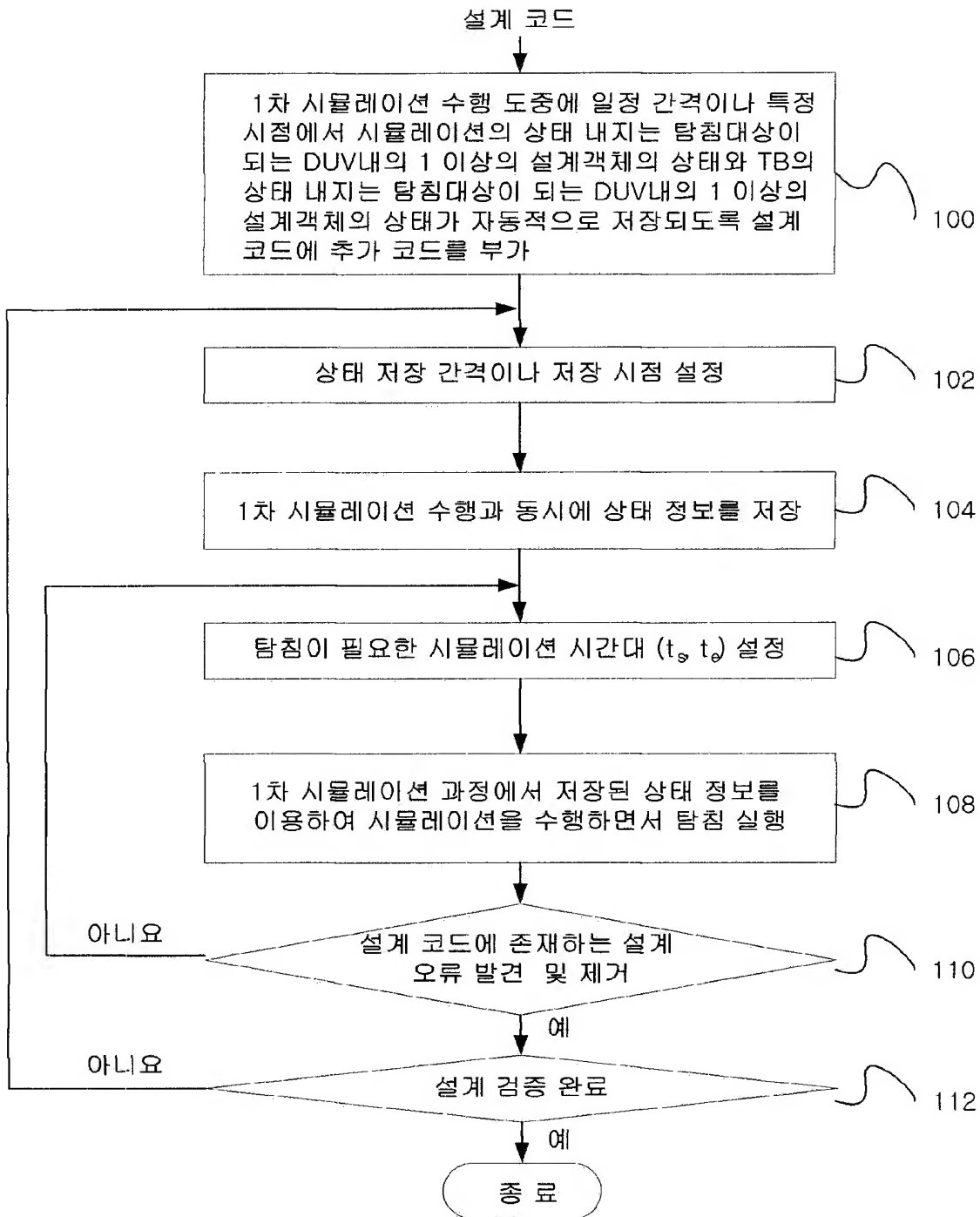
#### i) 1차 시뮬레이션



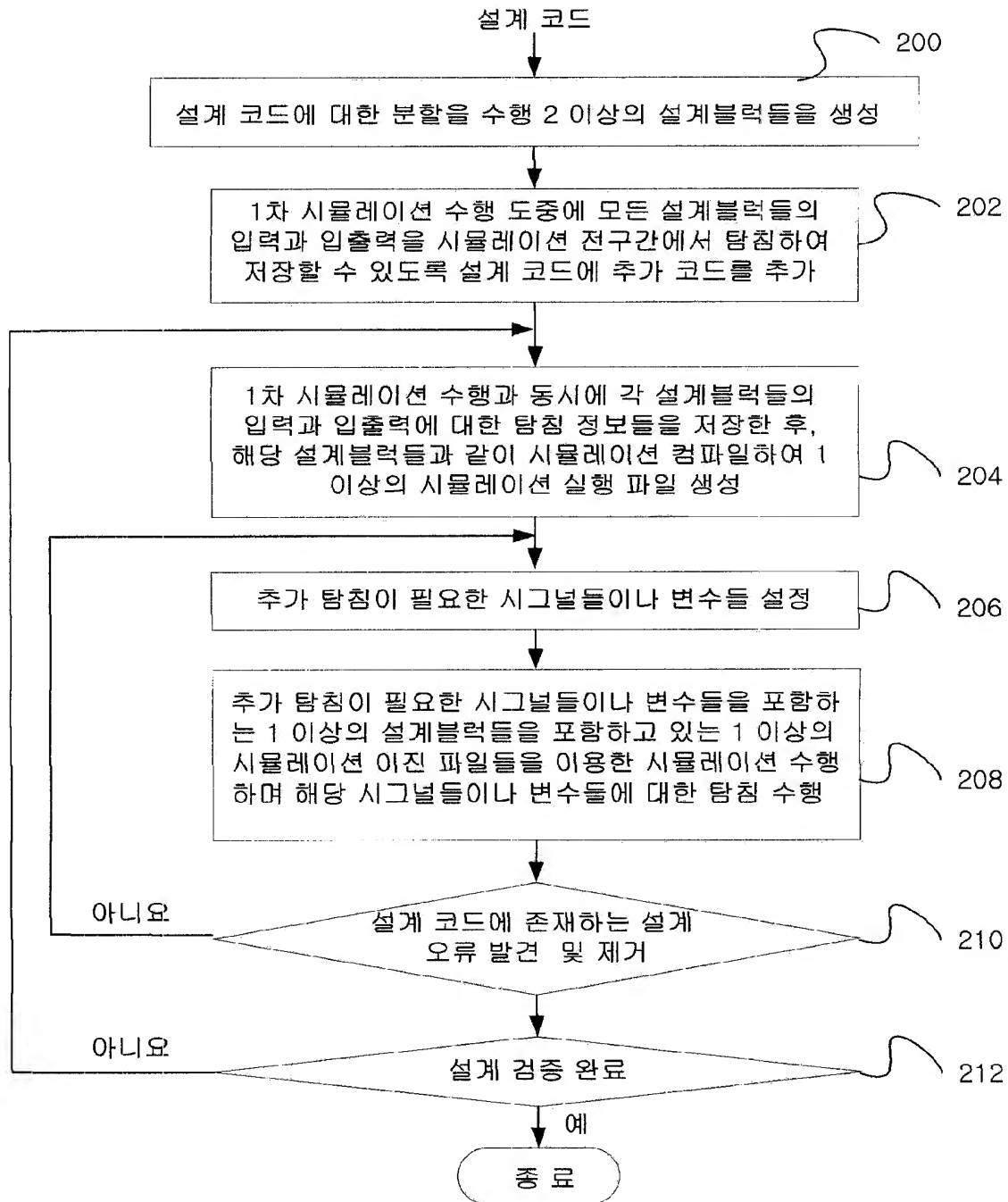
#### ii) 1차 이후의 시뮬레이션



【도 5a】



【도 5b】

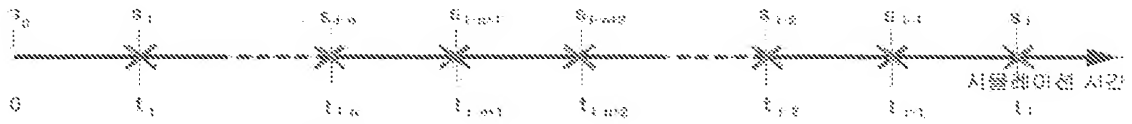




【도 6a】

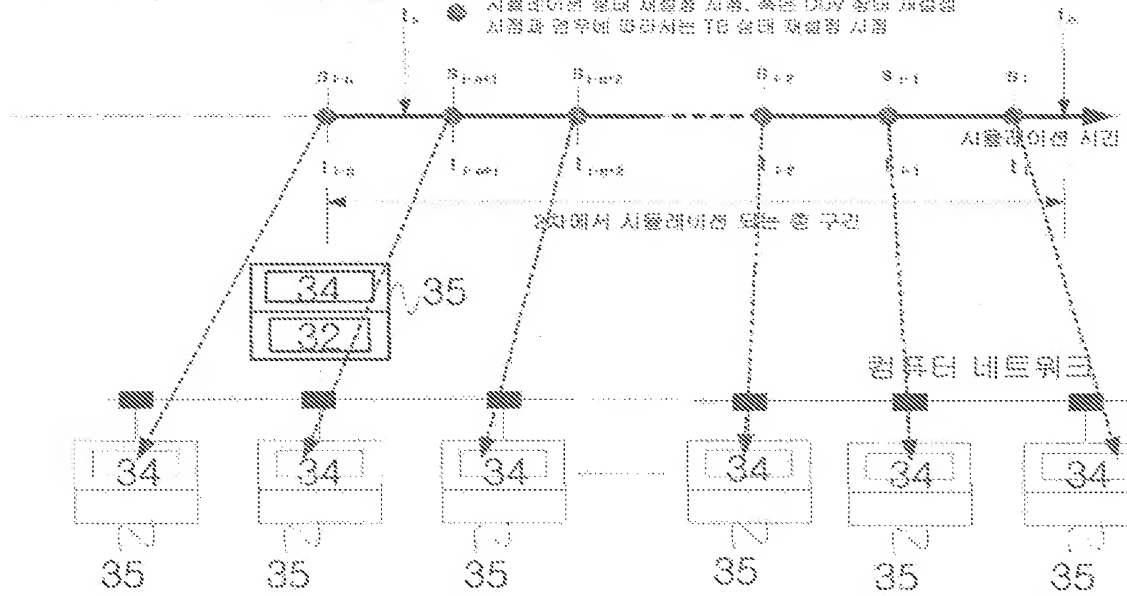
i) 1차 시뮬레이션

✕ 시뮬레이션 상태 저장 시점, ●은 DUV 상태 저장 시점과 경우에 따라서는 TB 상태 저장 시점

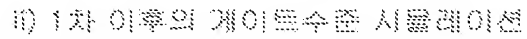


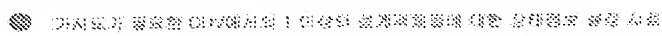
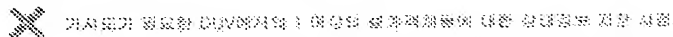
ii) 1차 이후의 시뮬레이션

● 시뮬레이션 상태 재설정 시점, ●은 DUV 상태 재설정 시점과 경우에 따라서는 TB 상태 재설정 시점



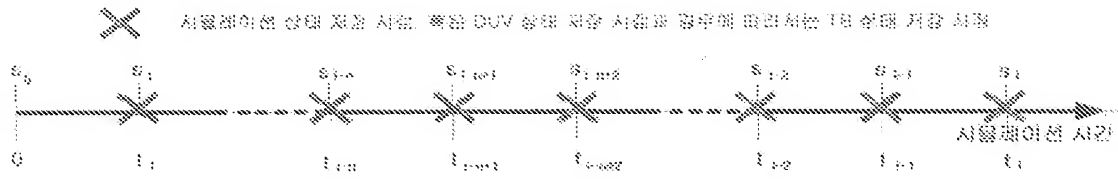
### 1) 1차 RTL 시뮬레이션



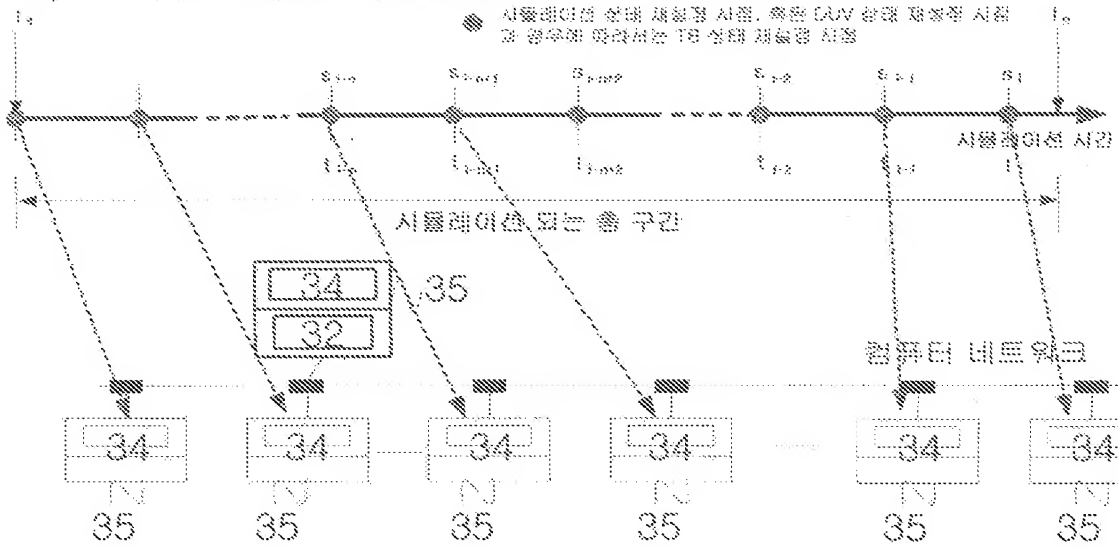


【도 6d】

i) 1차 시뮬레이션



ii) 1차 이후의 첫번째 병렬 시뮬레이션



iii) 1차 이후의 두번째 병렬 시뮬레이션

